GENETIC ALGORITHM BASED DIGITAL IIR FILTER DESIGN

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Michael James Schmitz

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

December 2004

Fargo, North Dakota

ABSTRACT

Schmitz, Michael James, M.S., Department of Electrical and Computer Engineering, College of Engineering and Architecture, North Dakota State University, December 2004. Genetic Algorithm Based Digital IIR Filter Design. Major Professor: Dr. Jacob Glower.

A method for designing a digital IIR filter with arbitrary magnitude response using a modified genetic algorithm (GA) is presented. A GA that operates on a complex, continuous search space is constructed and optimized by statistically determining the abilities of commonly used genetic operators. Furthermore, a new genetic operator is presented; it combines crossover and adaptive mutation to improve the convergence rate and solution quality of the GA.

A customized application layer, called the Filter Design Algorithm (FDA), has been developed for the optimized GA to handle the specific format and properties of the filter design problem. These requirements include a method for mapping a filter into the GA, evaluating the fitness of a filter, creating an initial population of filters, and ensuring that all filters are realizable.

ACKNOWLEDGMENTS

A great deal of time and effort, not all of it necessarily well spent, went into the development of this research. First, I want to apologize to everyone that had to listen to me babble about genetic algorithms and all of the "cool" plots that I made. That is, unless, we were at the Turf. You should have known better than to hang out at the bar with a grad student after he spent all day doing research.

I want to thank Joel Jorgenson and Jake Glower for convincing me to investigate genetic algorithms. It has certainly proved to be an interesting topic. All of this would not have been possible without the support of my friends, family, coworkers, and professors. You know who you are. Finally, I want to thank Packet Digital, started by Joel Jorgenson, for believing in me enough to hire me as employee #3 and encouraging me to continue my education by pursuing a Ph.D. in Electrical Engineering. I may just be in college forever.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

As the power and speed of current digital signal processor (DSP) hardware increases, the role of digital filters in fields such as communications and signal processing has become increasingly important. Digital filters can be applied to an almost endless variety of applications, including signal manipulation, noise cancellation, and channel equalization. Likewise, the frequency response required of a filter can be as varied and unique as its purpose. Therefore, efficient design strategies are necessary to resolve the digital filter transfer function needed for useful implementation.

Several design algorithms exist for finite-impulse-response (FIR) filters that enable relatively simple use of the filters in common practice. For example, the frequency weighted least squares (FWLS) technique enables an FIR filter of a given order to be designed according to a desired frequency response [1]. The resulting design equation has a closed form solution resulting in a very fast, accurate, and precise result. The design of infinite-impulse-response (IIR) filters, on the other hand, has shown to be more difficult. IIR filters are often preferred to FIR filters because of excellent magnitude response characteristics, especially when high attenuation or sharp transition bands are desired [2].

The digital IIR filter design task can be approached like any other optimization problem. Many methods of function optimization have been extensively studied and implemented, but the more traditional methods, such as calculus based hill-climbing, enumerative, and random approaches, often suffer from debilitating consequences [3]. To counter these pitfalls, evolutionary strategies for optimization have been gaining attention in recent years [4]. The main players in this evolutionary field are simulated annealing (SA) and genetic algorithms (GA), the latter being the focus of this thesis.

The concept of evolution was popularized by Charles Darwin in the early 1900s

[5], however, it took about another 80 years before John Holland definitively applied evolution to the function optimization process and produced the basic GA template still in use today [4]. So what makes the GA so great? One answer is its robustness. Its underlying architecture is designed to support a wide variety of optimization problems without the need of supplemental data. In the calculus based optimization approach, the function derivative is required to proceed in the optimal direction, but this information is not always available. Furthermore, enumerative techniques and random walks suffer from inefficiency derived from the need to evaluate the function at all or almost all points in the search space with no regard for the knowledge that is obtained from the evaluation process [3]. The GA, however, is a parallel optimization technique that relies on the basics of evolution for optimizing a group of solutions at once. This makes it better suited for locating the global optimum of an optimization problem, and the evolutionary approach efficiently drives the entire group of solutions towards this result.

1.1. Thesis Topic

Boundless applications can benefit from the GA's inherent robustness, but not directly. The GA is merely a framework underneath the actual algorithm required for optimization, and it must be customized for a given application. This leads to the question: "How should a genetic algorithm be customized for the design and optimization problem of a digital IIR filter with an arbitrary magnitude response?" This thesis attempts to answer this question.

1.2. Previous Work

Several methods for digital IIR filter design have been proposed and implemented. An application where filter design is important is an adaptive filter system that tracks a specified response. The least mean square (LMS) algorithm is traditionally used for adaptive FIR design, but it does not work well in the IIR

case due to a multi-modal mean square error (MSE) objective function. To rectify this, a global least mean square (GLMS) algorithm has been proposed to allow a global minimum search. This is accomplished by creating a convex objective function by "smoothing" the MSE function. As the algorithm converges closer to the global minimum, the amount of "smoothing" is reduced until it returns to the original objective function. This technique, similar to simulated annealing, is called stochastic approximation with convolution smoothing (SAS) and is realized by convolving the objective function with a noise probability density function. When combined with the LMS algorithm, it has shown success in adaptive IIR filtering [6].

Another approach for adaptive IIR filter design is also an extension of the LMS algorithm. The LMS algorithm is applied to multiple filters of different initial conditions to help reduce the probability of convergence to a local minimum. Whenever the rate of convergence slows or a filter becomes unstable, an embedded evolutionary computation is used to agitate the previous filter coefficients. This approach benefits from the directed search of the LMS algorithm and the ability to recover from instability, but simply implementing parallel optimization functions that all suffer from premature convergence to local minimum does not necessarily imply convergence to the global minimum. Furthermore, costly algorithms are required by this method to determine when transfer function coefficients lead to filter instability [7].

Doing away with the LMS algorithm altogether, a hybrid approach using both an SA and a GA has been proposed for adaptive IIR filters. A GA is used because of its implicit parallelism and robustness. However, if not properly tuned, a GA can lose critical components of good solutions and thus lead to premature convergence. An SA is used to counter this by incorporating it into the mutation strategy. The filter parameters are perturbed during mutation by appending random noise in an

amount dictated by the decay rate of the SA. Simulated results show improvements in global optimization compared to the LMS algorithm, pure SA, and pure GA, but the proposed technique has yet be actually applied to the IIR design problem for investigation of abilities [8].

Several methods for digital IIR filter design based purely on a GA have been proposed. The main differences between them lie in the customization of the algorithm for use in the IIR design problem. In one example, the focus of the algorithm is to produce an IIR filter with near linear phase. The method of generating a sample population, performing crossover and mutation, and evaluating the results are all tailored to this goal [9]. However, exact algorithm construction and execution is generally not provided. This makes usage and implementation difficult.

1.3. Goals and Motivation

Extensive work and thought by many has gone into analyzing the performance and abilities of the GA, but it is by no means fully understood. The majority of past GA study has concentrated on optimization problems with discrete parameters and search spaces. Only in roughly the last decade has GA analysis focused on the optimization of problems with continuous search spaces. During this time, many techniques and rules of thumb for constructing a continuous genetic algorithm (CGA) have been proposed.

Lessons learned from CGA examination and implementation are important to the filter design problem. The design of a digital IIR filter with an arbitrary magnitude response necessitates an algorithm capable of optimization in a continuous search space. This is due to the continuous valued coefficients or roots of a filter transfer function.

There are several goals for this thesis. First, traditional GA theory and common CGA practices will be presented. Reasoning for several existing CGA operators will be

given, and a new CGA crossover operator will be proposed. Second, the performance and abilities of the CGA will be characterized by applying it to the optimization of randomly generated multimodal objective functions. This will be done for several configurations of the CGA to determine the effect and interaction of the different CGA operators and parameters, such as selection, crossover, and mutation strategies. The empirical and statistical results from simulation will be used to intelligently select genetic operators that additively improve CGA optimization abilities.

Schemes for applying the CGA to the filter design problem will be presented. These include strategies for mapping a filter transfer function into the CGA population set, evaluating the quality of filters in the population compared to the desired magnitude response, and preserving or enforcing both required and desired digital IIR filter properties. The application of the CGA to the design problem of a digital IIR filter with arbitrary magnitude response results in the Filter Design Algorithm (FDA).

The FDA will be empirically analyzed by applying it to two types of design problems. First, the FDA will be used to design filters with a magnitude response with a known transfer function, such as a Butterworth bandpass filter. This is done to judge the optimization abilities of the FDA. Lastly, the FDA will be used to design a filter with an arbitrary magnitude response with an unknown transfer function to demonstrate its capabilities in otherwise difficult problems.

1.4. Organization

This thesis is organized as follows:

- Chapter 1. Introduction
- Chapter 2. Traditional GA Theory
- Chapter 3. Continuous GA Development (Statistical analysis of results is provided.)

- Chapter 4. Filter Design Algorithm (Five test cases are provided.)
- Chapter 5. Discussions
- Chapter 6. Future Work and Conclusions

CHAPTER 2. TRADITIONAL GA THEORY

2.1. Introduction

The focus of this chapter is to intoduce the concept of GA-based parameter optimization. Traditional Discrete Genetic Algorithm (DGA) analysis is presented along with its capabilities. Furthermore, the questions and requirements posed by a GA based digital filter design technique are outlined.

2.2. GA Descriptors

The modern GA was proposed by Holland [4] to understand adaptation in artificial systems. It is a powerful optimization scheme based upon an evolutionary computation technique that models genetics and natural selection. There are four main characteristics of the GA that set it apart from traditional optimization algorithms [3]. The GA

- operates on a representation of the problem function, not the function itself
- optimizes a set of candidate solutions rather than a single solution
- evaluates solutions with payoff and cost rules rather than supplementary knowledge
- uses probabilistic, rather than deterministic, transition rules.

2.3. Natural Evolution

The GA analysis presented here is structured in a top-down approach beginning with a conceptual description of the genetics comprising natural evolution. Natural evolution can be described as an "invisible force" that controls species adaptation and, in most cases, directs species development towards superior fitness. Three important concepts of natural evolution theory are variation, selection, and reproduction. Genetic variation within a species population is necessary to provide enough

information and resources for evolution to proceed towards the desired result. A lack of variation can lead to sub-optimal species development. Selection is the method of choosing which individuals from a population should be used to build subsequent future generations. According to Darwin, "The preservation of favourable (sic) individual differences and variations, and the destruction of those which are injurious, I have called Natural Selection, or the Survival of the Fittest [5]." Selected individuals produce a new population set of offspring through reproduction. This can often be considered a two step process consisting of recombination and mutation. Recombination is the act of combining genetic material in some manner from two parent individuals to produce offspring. This allows desirable traits from separate individuals to be combined and passed on to the next generation of offspring. However, recombination is not always perfect. Genetic mutation during reproduction is descriptive of errors in recombination. Mutation refers to the altering of genetic material in the offspring and can result in the increase of variation in the overall genetic pool.

To aid in understanding, the genetic building blocks of natural evolution are introduced. Each cell of a biological organism contains a set of *chromosomes* which are subdivided into a set of *genes*. Each gene can take on a range of values or traits called *alleles*. The group of genes describing an organism is called a *genome*, and the specific alleles of the genes identify the organism's *genotype*. The genotype is the genetic fingerprint of an organism, but it can sometimes be difficult to interpret and analyze. Therefore, the smaller set of all characteristics outwardly visible in an organism is called the *phenotype*. The phenotype determines an organism's fitness, and thus, its chance of selection for reproduction.

2.4. Artificial Evolution

Artificial evolution is an extrapolation of natural evolution. In a problem

optimization sense, artificial evolution operates on a set of synthetic chromosomes indicative of possible solutions with the goal of directing the solution set towards the optimal result. Artificial evolution's parallel to survival of the fittest is found in the *fitness function*. The fitness function quantifies the phenotype of an individual by assigning it a fitness level. The fitness function is unique to the particular evolutionary problem, and its fitness output is used to probabilistically drive selection and reproduction. Future generations are created through the application of crossover (recombination) and mutation to selected chromosomes to generate offspring. The selection, crossover, and mutation methods incorporated are of great importance and are generally chosen to expedite the convergence of the population towards the ideal phenotype solution. This entire process of fitness evaluation, selection, crossover, and mutation is repeated until a suitable phenotype result is achieved.

This process is exemplified by the traditional Discrete Genetic Algorithm (DGA). As its name suggests, the DGA operates on a discrete set of chromosomes where, for sake of simplicity, the chromosomes are generally considered to be binary encoded as single-bit genes. A basic block diagram of the DGA is shown in Figure 1.

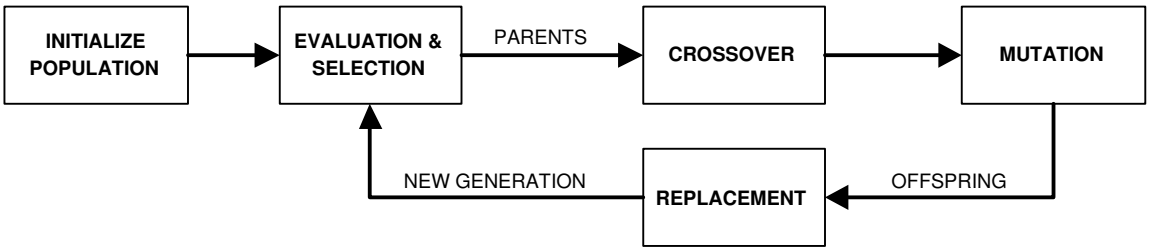

Figure 1. Traditional DGA block diagram.

The DGA has one major difference from natural evolution. Since the DGA operates on a simulated population in an artificial environment, the criteria for

selection, the amount of variation, and the frequency and degree of recombination and mutation can be adjusted by the user to achieve the desired optimization results. The most important requirement of the DGA is to balance exploration of the search space with the exploitation of available solution information. Exploration is necessary for the DGA to be a global optimization strategy, and exploitation is needed for the DGA to perform local searches and fine-tune solutions. The balance of the two is achieved through the selection, crossover, and mutation strategies chosen for the DGA.

2.5. Representation and Fitness

The DGA operates on a representation of the optimization problem function. This requires an encoding function

$$C : S \rightarrow X \tag{1}$$

to map the complete solution space S of the problem into the chromosome space X [10]. An individual chromosome (element) x_n, where n indicates the n^{th} sample in a set, is the encoded version of a particular solution with phenotype s_n and is related by

$$s_n = C(x_n). \tag{2}$$

The genotype of a chromosome is represented by

$$x_n = \{a_{n,1}, \cdots, a_{n,m}, \cdots, a_{n,M}\}, \tag{3}$$

where m identifies a gene location in x_n, and $a_{n,m}$ is the allele (vector) of the m^{th} gene. M is the total number of genes in x_n. In the DGA, an element is a binary encoded string with single-bit vectors.

The DGA operates on a time (generation) dependent population set of elements.

The population P at generation g is denoted

$$P(g) = \{x_1^g, \cdots, x_n^g, \cdots, x_N^g\}, \tag{4}$$

where N is the total number of elements in $P(g)$.

The fitness function $f(\cdot)$ is used to evaluate the fitness level of an element in the population in order to drive the search of the DGA. The fitness function is usually an objective or cost function, but anything will suffice as long as it is able to successfully quantify the quality of all possible phenotype solutions. The fitness function is dependent on the environment and application of the system that is undergoing adaptation, and it is the only source of information provided to the DGA for optimizing the population. The M-dimensional surface created by the fitness function is known as the fitness landscape. Every element x_n maps to a single point on this landscape. The object of the DGA is to direct the elements in $P(g)$ towards the global optimum of this surface.

The value of $f(\cdot)$ decreases as the fitness level increases. This may seem contradictory, and indeed this is the opposite of the traditional DGA interpretation, but it is the preferred definition for this thesis as it implies that a maximally fit element will have a fitness level of zero rather than infinity.

2.6. Selection

The purpose of selection is to drive $P(g)$ towards the most promising areas of S while still maintaining enough variation in $P(g)$ to prevent premature solution convergence. To comply with the accepted notion of survival of the fittest, any selection strategy chosen for the DGA must have a higher probability p_s of selecting the more fit elements of $P(g)$ to form the selection subset $P_s(g)$. The three most common methods of selection are *proportionate*, *rank-based*, and *truncation selection*, each with their own set of advantages and disadvantages.

2.6.1. Proportionate Selection

Proportionate selection was originally proposed by Holland and is the most popular form of selection. It operates just as its name implies in that the probability of selection is based on fitness proportions in the population. Let the *total fitness* F of population P be defined as

$$F = \sum_{n=1}^{N} \frac{1}{f\left(x_n\right)}, \tag{5}$$

where $f(x_n)$ is the fitness level of element x_n. Then, the probability of selecting the n^{th} element in P is

$$p_{s,n} = \frac{1}{F \times f(x_n)}, \tag{6}$$

assuming the reciprocal definition of fitness defined previously.

The following should be noted. Proportionate selection

- is a sampling with replacement strategy
- has a higher probability of selecting superior elements, but can still select elements with lesser fitness
- can fill the entire selection subset with the same element.

Drawbacks of this selection technique can be seen in highly diverse and near homogeneous populations. When P is highly diverse, it is possible for a few elements of P to have a large effect on selection. These elements can have a very high probability of selection and will thus dominate P_s, thereby leading to a loss of genetic information and prompting premature system convergence [11, 12]. On the other hand, if there is little variation in the fitness levels of P, then proportionate selection reduces to a simple random choice. It will no longer be able to direct the movement of the elements through the search space S, thus leading to slow system convergence [10].

2.6.2. Rank-Based Selection

Rank-based selection is fashioned to combat the pitfalls of proportionate selection. A major difference between the two is seen in the importance of $f(\cdot)$, where the exact value plays a minor role in calculating p_s in rank-based selection. Rather, rank-based selection is performed by ranking the elements of P based on fitness. The least fit element is assigned a rank of r_1; the most fit element receives the rank r_N; and the remaining elements are distributed in between. Based on this, the equation for $p_{s,n}$ becomes

$$p_{s,n} = \frac{1}{N}\left[min + (max - min)\frac{r_n - 1}{N - 1}\right], \tag{7}$$

where the numerical value of r_n is equal to the value of the associated subscript, n. The variable min is the expected number of appearances in P_s of the element ranked r_1, and max is the expected number of appearances in P_s of the element ranked r_N. They are equal to

$$min = k \times p_{s,1} \tag{8}$$

$$max = k \times p_{s,N}, \tag{9}$$

where k is the total number of elements selected [10, 13]. Usually $k = N$. The values chosen for min and max dictate the selection pressure of rank-based selection, that of which is beyond the scope of this thesis.

Rank-based selection is sampling with replacement like proportionate selection but does not suffer from the same fitness diversity issues. Element ranking alleviates diversity dependency. A drawback of rank-based selection is that it does not conform to traditional GA theory, and it is, thus, harder to prove the strategy's abilities in the DGA environment. Furthermore, rank-based selection is computationally more

complex than proportionate selection.

2.6.3. Truncation Selection

Truncation selection is probably the simplest of all selection strategies short of a purely random method. Truncation selection creates P_s by truncating the top $T\%$ elements from P based on fitness. The best $T\%$ elements are selected, and the bottom $100-T\%$ are discarded [14, 15]. This strategy is similar to common breeding practices.

The following should be noted. Truncation selection

- is a sampling without replacement strategy
- will form a selection subset identical to P if $T = 100\%$
- is immune to fitness diversity issues seen in proportionate selection.

The quality of the selection subset $P_s(g)$ chosen with truncation selection is determined by the *response to selection* and *selection effectiveness* indicators. The response to selection $R(g)$ is calculated by

$$R(g) = \overline{f}(P(g+1)) - \overline{f}(P(g)), \tag{10}$$

where $\overline{f}(P(g))$ is the average fitness of $P(g)$. Selection effectiveness $S(g)$ is defined as

$$S(g) = \overline{f}(P_s(g)) - \overline{f}(P(g)), \tag{11}$$

where $\overline{f}(P_s(g))$ is the average fitness of the selection subset $P_s(g)$ [10].

2.7. Crossover

The crossover genetic operator redistributes genetic material within $P(g)$. It is generally the first genetic operator applied after selection and is often considered the most important genetic operator in the DGA. Crossover is executed by combining or

mixing two elements from P_s with probability p_c to form one or more offspring that encapsulate the combined genetic material. The goal is to generate new elements that are more fit than their parents, thereby contributing to the overall fitness and convergence of the population. The justification for crossover is supported by schema theory and the building block hypothesis, which will be discussed shortly.

Crossover operates on genetic information on a finite sized block basis, generally single element vectors. For example, a single-bit vector represents one crossover block in the DGA. As previously stated, the m^{th} vector or block of the n^{th} element of P will be written $a_{n,m}$. Several different crossover strategies have been integrated into the DGA, but the most common are *single-point*, *double-point*, and *uniform crossover*.

2.7.1. Single-Point Crossover

Assume two parent elements x_1 and x_2 with M vectors apiece are chosen with probability p_c from P_s to undergo crossover. The first step of single-point crossover is to randomly choose a crossover point k_1 from the set

$$K_1 = \{k_1 \in \mathbb{Z} : 1 \leq k_1 \leq M - 1\}, \tag{12}$$

where k_1 identifies a point in an element's vector set to split the set into two vector subsets. It is applied to both parent elements using the same value for k_1 in each. The second step in single-point crossover it to swap one vector subset from the same location in each parent and recombine it with the existing vector subset in the other parent to create two offspring elements $x_1^{'}$ and $x_2^{'}$ as follows

$$x_1^{'} = \{a_{1,1}^{'}, \cdots, a_{1,M}^{'}\} = \{a_{1,1}, \cdots, a_{1,k_1}, a_{2,k_1+1}, \cdots, a_{2,M}\} \tag{13}$$

$$x_2^{'} = \{a_{2,1}^{'}, \cdots, a_{2,M}^{'}\} = \{a_{2,1}, \cdots, a_{2,k_1}, a_{1,k_1+1}, \cdots, a_{1,M}\} \tag{14}$$

2.7.2. Double-Point Crossover

Double-point is very similar to single-point crossover, but, as its name implies, two crossover points are used instead of only one. The two crossover points k_1 and k_2 are chosen at random from the sets

$$K_1 = \{k_1 \in \mathbb{Z} : 1 \leq k_1 \leq M-2\} \tag{15}$$

$$K_2 = \{k_2 \in \mathbb{Z} : k_1+1 \leq k_2 \leq M-1\}. \tag{16}$$

The offspring $x_1^{'}$ and $x_2^{'}$ are created from parents x_1 and x_2 by

$$x_1^{'} = \{a_{1,1}^{'}, \cdots, a_{1,M}^{'}\} = \{a_{1,1}, \cdots, a_{1,k_1}, a_{2,k_1+1}, \cdots, a_{2,k_2}, a_{1,k_2+1}, \cdots, a_{1,M}\} \tag{17}$$

$$x_2^{'} = \{a_{2,1}^{'}, \cdots, a_{2,M}^{'}\} = \{a_{2,1}, \cdots, a_{2,k_1}, a_{1,k_1+1}, \cdots, a_{1,k_2}, a_{2,k_2+1}, \cdots, a_{2,M}\} \tag{18}$$

Double-point crossover has empirically shown to have an improved ability to progress the population towards the optimal solution compared to single-point. A suggested reason for this is that double-point crossover is less likely to disrupt schemata with a larger defining length $\delta(H)$ [12]. This will become more evident shortly during the discussion of schema theory.

2.7.3. Uniform Crossover

Uniform crossover, sometimes called n-point crossover, exchanges genetic material in a slightly different manner. Instead of swapping vector subsets created by crossover points, each vector in the offspring has an equal probability of coming from either parent element, and the choices made for all vectors are independent of one another. With parents x_1 and x_2, the vectors for $x_1^{'}$ are chosen with

$$a_{1,m}^{'} = \{a_{1,m}\} \text{ or } \{a_{2,m}\}, \tag{19}$$

where $a_{1,m}$ or $a_{2,m}$ are chosen with uniform probability and $a'_{1,m}$ is the m^{th} vector of x'_1. All M vectors are independently chosen in the same fashion. Offspring x'_2 is generated with the opposite vector choices used to form x'_1.

Syswerda has shown that uniform crossover is more effective than single and double-point crossover in most DGA applications. Uniform crossover is more disruptive to the population, but this also makes it is better suited for exploring the search space S [16].

2.8. Mutation

During the optimization and convergence process it is sometimes necessary to remove desirable genetic material from the population to overcome local optima. Furthermore, there is no guarantee that all necessary optimal genetic information appears in the population at a given generation. Therefore, the mutation operator has been developed to introduce new or lost genetic material into the population. Mutation is performed by probabilistically modifying vector values of offspring elements. This enables exploration into areas of the search space S that could not otherwise be reached through crossover of the current set of elements [4].

The probability of mutating a given vector in an offspring element is p_m. This probability has a large effect on the ratio of exploration to exploitation in the DGA. The greater the probability, the further S will be explored and the longer it will take for $P(g)$ to converge, if at all.

Bit-flipping mutation is generally used in the DGA where elements are encoded with single-bit vectors. This is the simplest form of mutation and is realized by inverting the bit vectors of an element with probability p_m. It is important to note that each vector is mutated independent of the others [4, 17]. As a rule of thumb,

the lower bound for the optimal probability of bit-flipping mutation is

$$p_m = \frac{1}{M}, \tag{20}$$

where M is the number of vectors in the element undergoing mutation. This may seem rather high, but it has been experimentally supported in natural evolution by forcing higher mutation rates in animals by exposing them to radiation [14].

2.9. Other Genetic Operators

Additional genetic operators for the DGA exist such as multiple parent crossover and incest prevention, but they are generally not implemented due to added complexity. The DGA has also shown to be robust and perform well without them. Incest prevention, however, is an interesting concept that has shown to help improve the quality of the solution generated by the DGA. Based on traditional genetic incest, one method of incest prevention prevents elements in the population that are ancestors or siblings from mating [18]. The depth in the family tree that this can be enforced is dependent on the size of the population since, at some level, all elements could be related. Another incest prevention method allows elements to mate only if the hamming distance between the two exceeds a specified threshold [19]. The threshold is initially set to the average hamming distance of the population and is slowly decreased as the population converges. The goal of this technique is to prevent overly similar elements from mating and, thus, forming similar offspring which do not aid in exploration of the search space. Both of these methods have been shown to reduce duplicate elements in the population and increase the diversity of the genetic pool.

2.10. Replacement

After all genetic operators have been applied to $P_s(g)$, the resulting subset $P_s^{'}(g)$ must be melded with $P(g)$ to form $P(g+1)$. This requires a *replacement strategy*. The

particular replacement strategy chosen largely depends on the number of elements $N_s^{'}$ in $P_s^{'}$ compared to the number of elements N in P. The most common practice is to have *total replacement* as in

$$\left[P(g+1) = P_s^{'}(g)\right] \Leftrightarrow \left[N = N_s^{'}\right] \tag{21}$$

as this closely models traditional breeding practices [4]. Other possible methods include *proportionate*, *rank-based*, or *truncation replacement* if $N_s^{'} < N$. These replacement strategies operate in similar fashion to their selection counterparts except that lesser fit elements are favored for replacement. Whichever method is chosen, it is important to ensure the N remains constant for all values of g to maintain a constant population size.

Another replacement method functions by having the offspring $P_s^{'}$ replace the selected parents P_s. Careful attention must be paid, however, if P_s was formed by sampling P with replacement as some elements in P may be identical. Furthermore, studies of this method have shown that it is advantageous to have offspring elements replace their parents only if the offspring express superior fitness [20].

2.11. Schema Theory

The optimization abilities and robustness of the DGA is supported by *schema theory* introduced by Holland [4] and added to by Goldberg [21, 3]. Rather than focusing on the vectors or elements in particular, schema theory is based on the similarities between elements in the population. Holland defines a schema, with notation H, to be a template that identifies the similarities between binary encoded elements with single-bit vectors. To do this, the symbol $\star$ meaning "don't care" was added to the binary alphabet of 0 and 1. For example, the schema $H = \{\star 1 \star \star 0\}$ is a similarity template that applies to all elements that have a 1 in the second position, a 0 in the fifth position, and either 0 or 1 in any position marked with a $\star$. A specific

schema H applies to 2^n elements, where n is the number of $\star$'s that appear in H.

For further insight consider the elements $A = \{10011\}$ and $B = \{00101\}$. The schemata $\{\star 0 \star\star\}$, $\{\star\star\star 1\}$, and $\{\star 0 \star\star 1\}$ apply to the elements A and B, but it should be noted that the schema $\{\star 0 \star\star 1\}$ is more specific and applies to a smaller overall set of possible elements. The specificity of schema H is $o(H)$ and is equal to the number of fixed positions in H, an example being $o(\{\star 0 \star\star 1\}) = 2$. The order of a schema is also important when discussing schema theory. The order of schema H is $\delta(H)$ and is equal to the distance between a schema's outermost defining positions. For example, $\delta(\{\star 0 \star\star 1\}) = 3$ and $\delta(\{\star 0 \star\star\star\}) = 0$.

The effects of recombination on the expected number of schemata in $P(g)$ with N elements is relatively easy to determine [21, 3]. The form of the following equations differ slightly from Goldberg's since a reciprocal definition of fitness is used in this thesis. To begin, let $m(H, g)$ equal the number of examples of a particular schema H that appear in $P(g)$. Using proportionate selection to produce a new population $P(g+1)$ of size N, the expected number of schema H in $P(g+1)$ is $m(H, g+1)$ given by the equation

$$m(H, g+1) = m(H, g)\frac{\overline{f}\,(P(g))}{\overline{f}\,(H(g))}. \tag{22}$$

$\overline{f}\,(H(g))$ is the average fitness of all elements in $P(g)$ containing schema H, and $\overline{f}\,(P(g))$ is the average fitness of all N elements in $P(g)$. Therefore, schemata that are more fit than the population average, $\overline{f}(H(g)) < \overline{f}(P(g))$, receive increasing samples in future generations, while less fit schemata receive decreasing samples.

Single-point crossover destroys schemata with a certain probability dependent on the proportional relationship between the defining length of the schemata and the length M of the elements. As $\delta(H)$ increases for a fixed M, the likelihood of the crossover point falling between the extreme fixed positions of H increases. If this happens, H will be destroyed assuming that the two parent elements under

examination are not identical at fixed positions of H. Because the survival of H depends on the crossover point falling outside the defining length, the probability of schemata surviving single-point crossover is

$$p_{sc} = 1 - \frac{\delta(H)}{M-1}. \tag{23}$$

If crossover is performed with probability p_c, then

$$p_{sc} \geq 1 - p_c \frac{\delta(H)}{M-1}. \tag{24}$$

In order for schemata to survive simple bit-flipping mutation occurring with probability p_m, all defining positions of the schemata must survive. The probability of a single vector surviving mutation is $(1-p_m)$. Since each mutation is independent and all of the $o(H)$ positions must survive, the overall probability of H surviving mutation is

$$p_{sm} = (1-p_m)^{o(H)} \simeq 1 - p_m o(H) \text{ for } p_m \ll 1. \tag{25}$$

Combining equations (22), (24), and (25) reveals the complete equation for schemata survival in the traditional DGA with proportionate selection, single-point crossover, and bit-flipping mutation.

$$m(H, g+1) \geq m(H, g) \frac{\overline{f}(P(g))}{\overline{f}(H(g))} \left\{ 1 - p_c \frac{\delta(H)}{M-1} - p_m o(H) \right\} \tag{26}$$

Analysis of (26) shows that short, low order schemata with above average fitness have a higher probability of survival and are given increasing occurrences in future generations. The remaining schemata receive fewer, hence, survival of the fittest.

Do fit schemata lead to fit individuals? In order for the DGA to be effective, the necessary answer is yes. However, it is hard to prove or quantize this claim.

The *building block hypothesis* proposed by Goldberg [3, 21] states that a simple DGA will indeed combine the above average, short, low order schemata (building blocks) to form fit individuals. Support for his hypothesis is shown by the success of evolution in nature. Despite this, Goldberg has shown that it is possible for a problem to deceive a DGA when the building blocks do not lead to near optimal solutions. An example of this would be a function with a global optimum solution surrounded by extreme poor solutions. Therefore, schema theory on its own is not enough to guarantee convergence in arbitrary problems, but it does show why it often occurs.

A more complex GA makes schema theory difficult to reckon. Complex alphabets, multi-point crossover strategies, and new mutation strategies change the specifics of the theory, but the general claim has shown to still hold true in implementation.

2.12. DGA-Based Filter Design

At first glance, the DGA appears to be a suitable optimization technique for the digital IIR filter design problem. After all, a discrete method should be an ideal approach to a digital problem. However, the nature of digital IIR filter design poses several problems for the DGA. A digital IIR filter can be defined with sets of recursive difference equations of the form

$$y[n] = -b_1 y[n-1] - b_2 y[n-2] - \cdots - b_i y[n-i] - \cdots - b_I y[n-I] \quad (27)$$

$$+c_0 x[n] + c_1 x[n-1] + \cdots + c_j x[n-j] + \cdots + c_J x[n-J], \quad (28)$$

where $x[n]$ is the digital input and $y[n]$ is the digital output of the filter. Examination of this equation reveals that the values of all output coefficients b_i, all input coefficients c_j, and the number of output and input samples I and J used in the response calculation must be intelligently chosen to achieve a desired digital IIR filter response. Ignoring I and J, it is, therefore, necessary for the DGA to optimize the set of values

for b_i and c_j to design a filter for a particular response. Herein lies the problem. Coefficients b_i and c_j represent a continuous set of parameters not suited for the discrete optimization of the DGA. Before a digital IIR filter can be designed with a GA, two questions must be asked: How should continuous parameters be represented in binary encoded elements and vectors? And what are the most effective selection, crossover, and mutation strategies for dealing with continuous parameters?

This section has presented the fundamental concepts behind DGA-based parameter optimization. Although the DGA has shown to be an efficient and robust algorithm for many optimization problems, it has also shown to not be entirely suited for the digital IIR filter design problem. The next section serves to solve some pitfalls of the DGA by introducing a related optimization method, the Continuous Genetic Algorithm.

CHAPTER 3. CONTINUOUS GA DEVELOPMENT

3.1. Introduction

The DGA is tailored to the optimization of discrete parameter sets. Digital IIR filter design, however, requires the optimization of a continuous parameter set. This necessitates modifications to the DGA including a technique for representing continuous parameters in binary encoded elements and vectors and new crossover and mutation strategies must be implemented to deal with continuous rather than discrete vectors.

In this chapter, several published options are presented and evaluated for resolving these issues. Furthermore, a new "hybrid" crossover technique is presented for consideration. For generalization, all proposed techniques are designed for complex continuous vectors. Finally, a randomly generated, mathematically generic, complex, continuous optimization problem is used to gather empirical results for several different selection, crossover, and mutation operator combinations for continuous encoded elements. Statistical analysis of the results is used to select the preferred combination of genetic operators to achieve superior fitness in the GA. The final algorithm presented for optimizing continuous problems is called the Continuous Genetic Algorithm (CGA).

3.2. Continuous Vector Encoding

An encoding function C capable of representing a continuous search space S in the element space X is required by the CGA such that

$$C : \mathbb{R} \rightarrow X. \tag{29}$$

It is impossible to exactly encode $\mathbb{R}$ in a binary format to allow processing in a digital environment. Instead, a comprise between representation accuracy and efficiency

must be reached.

A simple extension from DGA encoding is to try multiple-bit vectors, but this technique can suffer from poor vector range and resolution. The required vector length can quickly become unmanageable depending on the size of S.

More complex encoding functions using extended alphabets have taken longer to reach the mainstream of GA theory because of difficult examination and qualification. However, mathematical analysis [22] and empirical data [23, 24] have shown that using a larger encoding alphabet in a GA can improve optimization and convergence results when compared to the traditional single-bit vector encoding method. Specifically, a single precision floating-point alphabet has shown to be a suitable encoding scheme for the GA.

With that in mind, vectors in the CGA are encoded as double precision floating-point complex numbers. Complex vectors are used to generalize the CGA and make it easier to optimize problems in the complex domain, such as the filter design problem. Double precision numbers are used to increase the precision of the CGA results.

3.3. Crossover

The discrete crossover strategies of the DGA were developed for binary encoded elements with single-bit vectors, and they have shown to be effective in this environment. These methods, however, are not completely suited for elements with complex, continuous vectors. Therefore, a new set of crossover tactics are necessary for the CGA to be truly successful.

3.3.1. Discrete Crossover

Uniform crossover was initially developed for the DGA, but it still has some applicability to the CGA. Single and double-point crossover are not considered here since they have already shown to be inferior to uniform crossover, and it is reasonable to assume that this will remain true for the CGA. The equivalent form of uniform

crossover in the CGA realm is rightfully named *discrete crossover*.

In the case of binary encoded elements with single-bit vectors, discrete crossover has the potential to explore the entire binary search space S. This is accomplished through the exchange of genetic material, provided that both a '0' and '1' exist somewhere in P for every possible vector position in an element. This is not a difficult requirement to meet, especially when the elements in P are fairly diverse. It is even possible to achieve this with only two elements that are logical bit inverses of one another.

With a continuous search space, it is not reasonable to assume that a finite sized population of elements can fully explore S with discrete crossover alone. This is because the near infinite possible values of a continuous vector cannot all appear in the population at one time. Discrete crossover can still swap genetic material between continuous elements, but the search ability is limited to the restricted set of information appearing in P. This leaves mutation as the only means of exploring the otherwise unreachable areas thereby placing more importance on mutation in the CGA when discrete crossover is implemented [17, 10]. Despite this, discrete crossover will be tested in the CGA, if only to establish a performance baseline for the other crossover plans.

3.3.2. Intermediate Crossover

Rather than swapping element vectors as in discrete crossover, *intermediate crossover* merges parent vectors together to form blended offspring. Intermediate crossover is capable of generating offspring vectors that do not already exist in P, thereby removing the requirement that all obtainable vector values must exist somewhere in P in order to fully explore S.

Two existing intermediate crossover methods are considered for the CGA, the first being *average crossover*. Average crossover is a deterministic plan that creates

a single offspring element by averaging together two parent elements on the complex vector level [10, 17]. Assuming the parents are x_1 and x_2, the offspring $x_1^{'}$ is calculated according to

$$x_1^{'} = \{a_{1,1}^{'}, \cdots, a_{1,M}^{'}\} = \left\{ \frac{a_{1,1} + a_{2,1}}{2}, \cdots, \frac{a_{1,m} + a_{2,m}}{2}, \cdots, \frac{a_{1,M} + a_{2,M}}{2} \right\}. \quad (30)$$

One consequence of average crossover is that it can only create one unique offspring per two parents. This means that average crossover must be applied to more pairings of parents to achieve the same number of offspring as other techniques.

The second intermediate crossover strategy considered is *constant crossover* and is an example of a probabilistic method. In constant crossover, two offspring vector values $a_{1,m}^{'}$ and $a_{2,m}^{'}$ are chosen with a constant probability density function (PDF) and are located such that they fall on the line segment connecting the two parent vectors $a_{1,m}$ and $a_{2,m}$ in the complex plane [10, 17]. The equations for calculating the offspring vectors are

$$\begin{aligned} a_{1,m}^{'} &= (1 - X_1)\,\Re\{a_{1,m}\} + (X_1)\,\Re\{a_{2,m}\} \quad (31) \\ &\quad + j\left[(1 - X_1)\,\Im\{a_{1,m}\} + (X_1)\,\Im\{a_{2,m}\}\right] \\ a_{2,m}^{'} &= (1 - X_2)\,\Re\{a_{1,m}\} + (X_2)\,\Re\{a_{2,m}\} \quad (32) \\ &\quad + j\left[(1 - X_2)\,\Im\{a_{1,m}\} + (X_2)\,\Im\{a_{2,m}\}\right] \\ f_{X_i}(x) &= \begin{cases} 1, & 0 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}. \quad (33) \end{aligned}$$

A plot of the offspring vector distribution is shown in Figure 2.

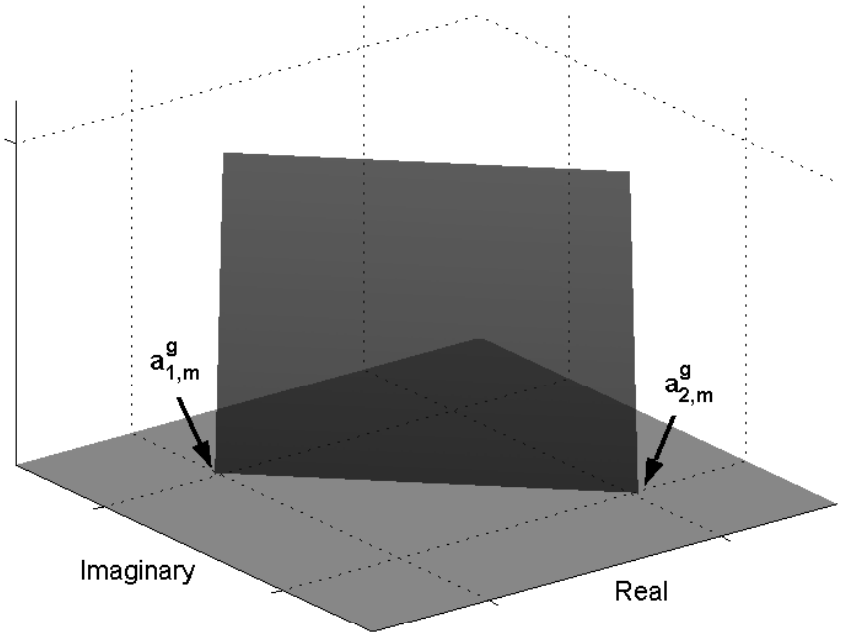

Figure 2. Constant crossover offspring vector distribution.

3.4. Mutation

Bit-flipping mutation is in no way suitable for the CGA. The continuous search space of the CGA is not conducive to a simple binary mutation strategy. Instead, three probabilistic mutation strategies called *uniform*, *normal*, and *breeder genetic algorithm (BGA)* mutation are considered for the CGA. These methods are designed for the complex continuous encoding methods used in the CGA.

Regardless of the strategy used, continuous mutation perturbs a continuous, complex vector by

$$a'_{n,m} = a_{n,m} + \lambda, \tag{34}$$

where λ is a randomly generated mutation factor based on the mutation range A.

A controlled mutation range is necessary to limit the effective change caused to an element by mutation. Furthermore, a given vector is still mutated with probability p_m. It is important to remember that, in general practice, the mutation operator is applied to only the offspring elements generated by crossover.

3.4.1. Uniform Mutation

Uniform mutation randomly chooses a complex value for λ using a uniform joint PDF in the complex plane [15]. The domain of the joint PDF is circular in shape to maintain a constant maximum mutation range A in any direction in the plane. The equations for calculating λ with uniform mutation are

$$\lambda = Re^{j\Theta} \tag{35}$$

$$f_{R,\Theta}(r,\theta) = \begin{cases} \frac{1}{\pi r^2}, & 0 \le r \le A,\ 0 \le \theta < 2\pi \\ 0, & \text{otherwise} \end{cases} . \tag{36}$$

The joint PDF for uniform mutation is shown in Figure 3.

3.4.2. Normal Mutation

Normal mutation, on the other hand, uses a gaussian joint PDF to select λ. The range of mutation A dictates the standard deviation σ of the distribution [15]. This technique has the advantage over uniform mutation in that it does not have a step size restriction preventing its escape from local optima. Furthermore, the gaussian shape favors small mutation ranges forming a strong inheritance between parents and offspring. The equations for choosing λ with normal mutation are

$$\lambda = X + jY \tag{37}$$

$$f_{X,Y}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\left[\left(\frac{x}{\sigma}\right)^2 - \left(\frac{y}{\sigma}\right)^2\right]}, \tag{38}$$

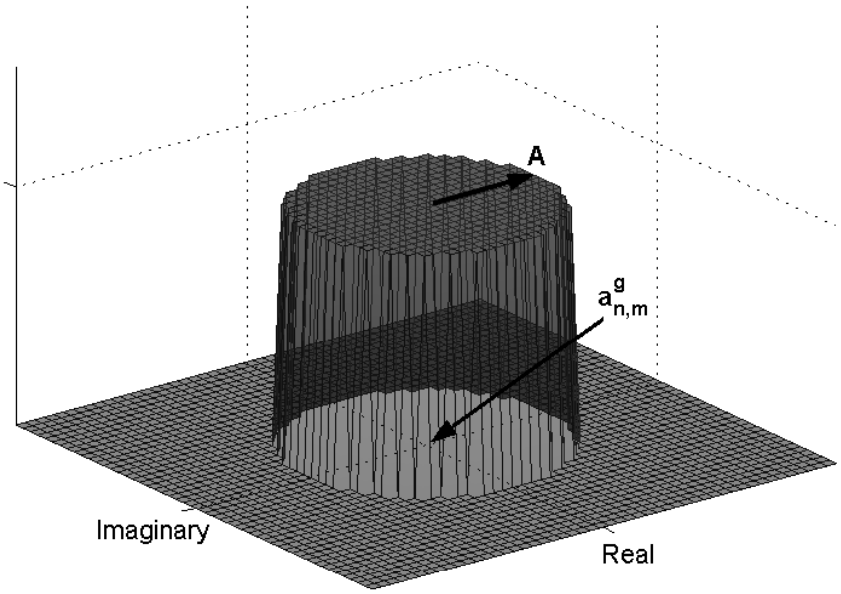

Figure 3. Uniform mutation joint PDF.

where

$$\sigma = \frac{A}{3}. \tag{39}$$

This produces a mutation distribution with approximately 99.7% of λ values in the same circularly defined mutation range as uniform mutation.

The joint PDF for normal mutation is shown in Figure 4.

3.4.3. BGA Mutation

Previous research has shown that a form of mutation range adaptation is required for uniform and normal mutation to be truly effective. This is because it is necessary to balance exploration of S with the exploitation of X. It is important for mutation to coarsely explore the search space S in the initial stages of the CGA

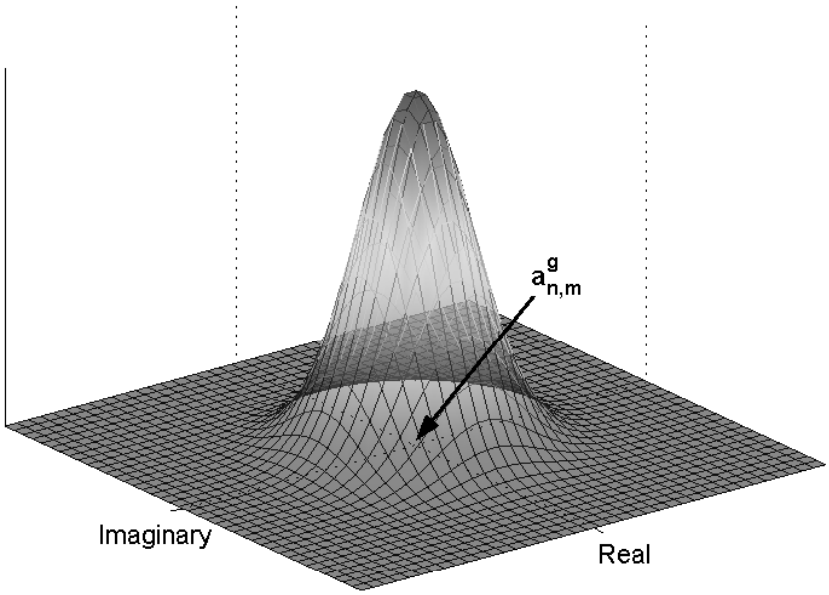

Figure 4. Normal mutation joint PDF.

to help steer $P(g)$ away from local optima. It is equally important for mutation to finely adjust vectors in the later stages of the CGA to help it remain in the area of the global optimum. Several methods for performing mutation range adaptation, such as using simulated annealing to adjust A based on g [8], have been proposed. However, this requires extra overhead and complexity in the algorithm.

The BGA mutation strategy was developed to remove the adaptation requirement from mutation with only a small decrease in efficiency compared to

uniform and normal mutation. Here, the method for selecting λ is

$$\lambda = X + jY \tag{40}$$

$$X = \pm A \sum_{i=1}^{16} \alpha_i 2^{-i} \tag{41}$$

$$Y = \pm A \sum_{i=1}^{16} \alpha_i 2^{-i}, \tag{42}$$

where α_i equals 1 with probability 1/16 and 0 otherwise, and α_i is independent of α_j when $i \neq j$. The value of X is generated independent of Y. The range of mutation for the BGA strategy is approximately $[-A, A]$ and $[-jA, jA]$. This method favors small numbers to allow for local tuning but can sometimes generate large numbers for solution exploration. BGA mutation has a resolution of 2^{-16} in the form given, but this can be modified as required by the optimization problem [15].

3.5. Hybrid Operator (Normal Crossover)

A hybrid operator that combines intermediate crossover and adaptive mutation is proposed in this thesis for the CGA. It is called *normal crossover* and is a probabilistic strategy similar to constant crossover. Rather than use a constant PDF, normal crossover uses a gaussian PDF to randomly select offspring vectors that lie along the collinear space defined by the parent vectors in the complex plane. The normal crossover equations for calculating the offspring vectors are

$$\begin{aligned} a'_{1,m} &= \Re\{\mu\} + (X_1)\,\Re\{a_{2,m} - a_{1,m}\} \\ &\quad + j\,[\Im\{\mu\} + (X_1)\,\Im\{a_{2,m} - a_{1,m}\}] \end{aligned} \tag{43}$$

$$\begin{aligned} a'_{2,m} &= \Re\{\mu\} + (X_2)\,\Re\{a_{2,m} - a_{1,m}\} \\ &\quad + j\,[\Im\{\mu\} + (X_2)\,\Im\{a_{2,m} - a_{1,m}\}] \end{aligned} \tag{44}$$

$$f_{X_i}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2}, \tag{45}$$

where

$$\mu = \frac{a_{1,m} + a_{2,m}}{2}. \tag{46}$$

The standard deviation σ is chosen so that there is an equally likely chance for $a'_{1,m}$ to lie in the region between $a_{1,m}$ and $a_{2,m}$ as there is for it to lie outside the two parent vectors. Assuming $a_{1,m} \leq a_{2,m}$ from a single-dimensional viewpoint and by using the standard normal CDF $\Phi(z)$ along with a table lookup to calculate σ,

$$\begin{aligned} P[a_{1,m} < a'_{1,m} \leq a_{2,m}] &= \Phi\left(\frac{a_{2,m} - \mu}{\sigma}\right) - \Phi\left(\frac{a_{1,m} - \mu}{\sigma}\right) && (47) \\ &= 2\Phi\left(\frac{a_{2,m} - a_{1,m}}{2\sigma}\right) - 1 && (48) \\ &= 0.5, && (49) \end{aligned}$$

the equation for the standard deviation is

$$\sigma \approx 0.741 \left| a_{2,m} - a_{1,m} \right|. \tag{50}$$

A plot of the offspring vector distribution is shown in Figure 5.

The hypothesis of normal crossover is that it can perform crossover and mutation in a single operator. Crossover is achieved through the offspring inheritance of genetic material from the parent elements dictated by the gaussian joint PDF defined by the parent vectors. This is a similar technique that has shown to be successful in constant crossover. Likewise, normal crossover is capable of mutation because the offspring are randomly selected within the region of the parent vectors. This, too, is similar to constant crossover, however, normal crossover is also capable of placing offspring outside the line segment defined by the two parents. This allows normal crossover to be effective on both concave and convex fitness landscapes whereas constant crossover is only effective in concave regions. The mutation adaptation factor is derived from

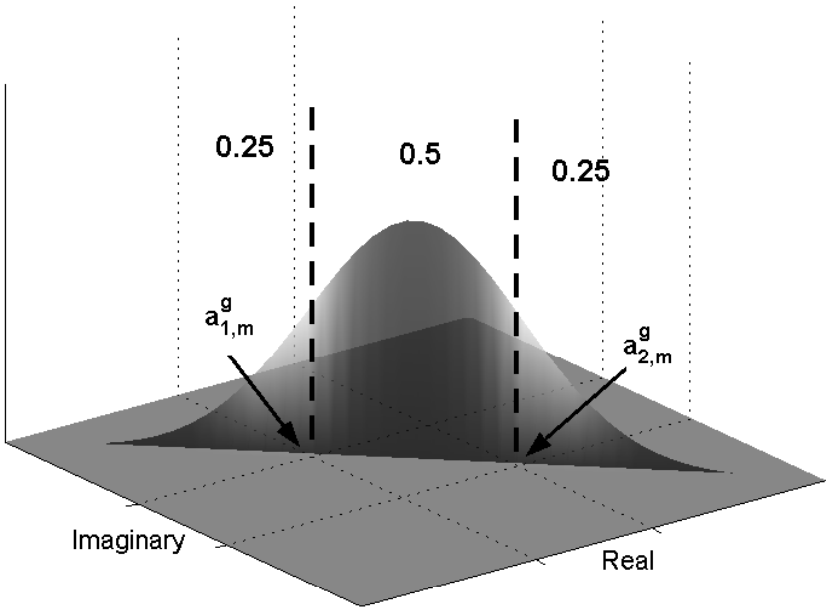

Figure 5. Normal crossover offspring vector distribution.

the calculation of σ. The standard deviation of the gaussian joint PDF is dependent on the distance between the parent vectors. Therefore, as the population converges, the distance between parent vectors will decrease and restrict the range of mutation. This feature causes a coarse search in a diverse population and a fine search in a converging population.

3.6. Complete CGA Form

The complete form of the CGA is shown in Figure 6. It is quite similar to the traditional DGA block diagram except for the use of continuous rather than discrete operations. For sake of completeness, the details of each stage of the algorithm are provided. All options considered for the genetic operators are listed. The new hybrid

operator, normal crossover, is classified as a crossover method that can be combined with an additional mutation method if desired.

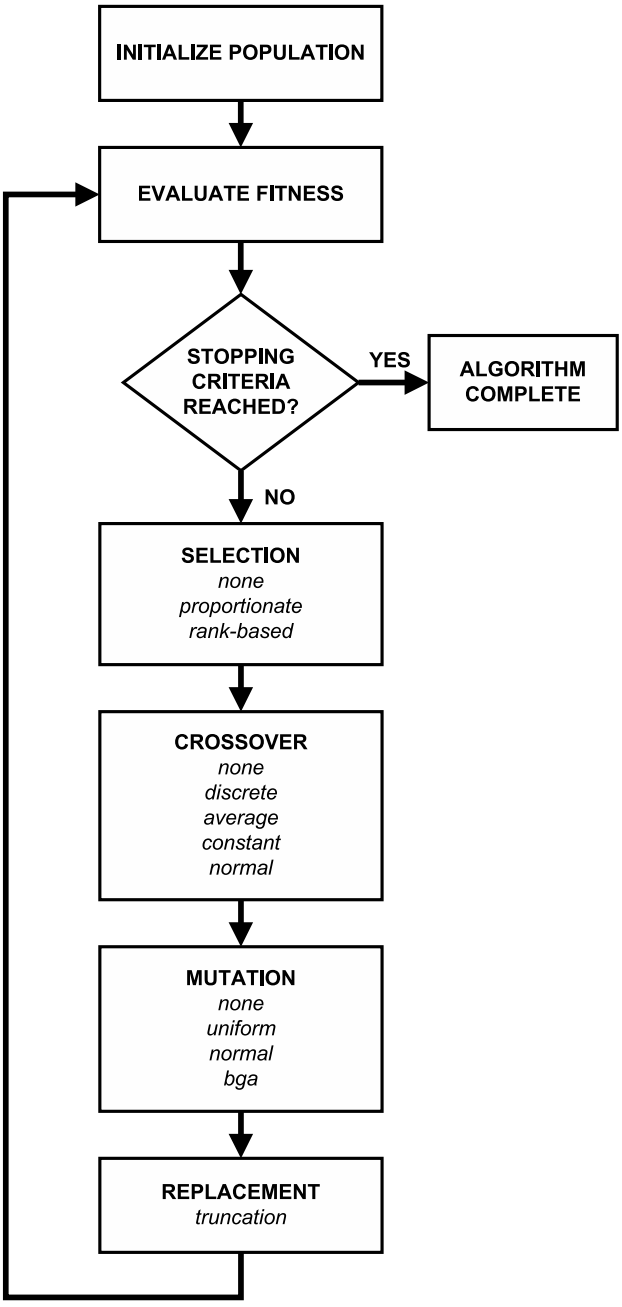

Figure 6. CGA block diagram.

- The initial population $P(0)$ is randomly generated by selecting double precision floating-point, complex numbers with uniform probability for all vectors $a^0_{n,m}$ from the chromosome space X that encodes the solution space S.

- Element fitness $f(x_n)$ is calculated with an objective function that is specific to the given application of the CGA with the requirement that a lower fitness value indicates a better solution.

- The CGA terminates when specific criteria are met. The two used in the CGA are called gen_max and fit_min. The algorithm stops when it has executed the number of generation specified by the value of gen_max regardless of the quality of solutions. This is to prevent the CGA from running forever. The algorithm can also stop when an element in the current population has a fitness value equal to or less than the value of fit_min. This saves time by stopping the CGA when a solution of adequate quality is achieved.

- A subset of elements is selected for recombination during selection. The size of the selection subset is the same as the population. Possible selection strategies are proportionate, rank-based, or none at all. When no selection is used, the selection subset is identical to the population set. Truncation selection is not considered here.

- Elements are chosen from the selection subset with probability p_c to become crossover parents. Possible crossover strategies are none, discrete, average, constant, or normal. When no crossover is used, parents selected for crossover become the offspring. When average crossover is used, the number of generated offspring is one-half the number of parents. All other techniques produce the same number of offspring as parents.

- Vector values of the offspring elements are perturbed with probability p_m.

Possible mutation methods are uniform, normal, BGA, or none at all. This step is skipped if no mutation is to be performed. Offspring elements from crossover are the only elements that can undergo mutation.

- The next generation $P(g+1)$ is created by adding the offspring elements into the current population of elements. This is done with a truncation technique that removes the worst elements from $P(g)$ based on fitness to make room for the offspring. The size of $P(g+1)$ remains the same as $P(g)$.

3.7. CGA Simulation

Multiple methods exist for selection, crossover, and mutation in the CGA. An empirical analysis of the CGA is performed for different combinations of these operators to quantify the performance and abilities of each. Generic, randomly generated test functions are used to help remove any objective function dependencies, thereby creating a generalized set of results. These results are statistically analyzed to select the best CGA configuration for complex, continuous parameter optimization.

3.7.1. Test Functions

A set of test functions is necessary for evaluating the performance and ability of the CGA. In his revealing Ph.D. dissertation, De Jong proposed a test suite of functions for evaluating the effectiveness of different GA configurations [25]. De Jong was careful to select functions that cover a breadth of function characteristics, including [3]

- continuous/discontinuous
- convex/nonconvex
- unimodal/multimodal
- quadratic/nonquadratic

- low-dimensionality/high-dimensionality
- deterministic/stochastic.

His test functions are helpful for controlled experimentation of the GA and are commonly used by others in the study of the GA for easy comparison of findings.

Spears, a student of De Jong, identified a weakness in De Jong's rationale [26]. The concern is that results obtained from standard empirical studies may not generalize beyond the applied test functions. This is seen when a new GA is fine-tuned to a particular set of test functions until it outperforms existing methods. Spears' proposed solution is to eliminate the ability to tune a GA to a particular set of problems. This is accomplished by using what Spears calls *test-problem generators* that randomly produce test problems that conform to a class of problems. This allows experimental results to be gathered over a randomly generated set of problems rather than a set of hand-selected problems.

The test-problem generator proposed by Spears is designed for a binary encoded GA. It produces a random multimodality test problem with the function

$$f(string) = \frac{1}{L} \max_{i=1}^{\mathbb{P}} \{L - \text{hamming}(string, Peak_i)\}, \tag{51}$$

where the number of peaks $\mathbb{P}$ can be easily controlled. The basic idea is this: A set of $\mathbb{P}$ L-bit strings are generated at random. These strings represent the locations of the peaks of the generated test function and are identified by $Peak_i$. To evaluate the fitness of a string, $f(string)$, using the randomly generated fitness surface, the first step is to locate the nearest peak in hamming space. The fitness of the string is then equal to the number of bits the string and the closest peak have in common, divided by the length L.

A modified version of (51) is needed for the CGA to account for the complex,

continuous encoding scheme and the reciprocal definition of fitness. Using notation standard to this thesis, the test-problem generator for the CGA is written as

$$f(x_n) = \frac{1}{M} \overset{\mathbb{P}}{\min_{i=1}} \left\{ \sum_{m=1}^{M} |a_{n,m} - Peak_{i,m}| \right\}. \tag{52}$$

This function operates on the vector level with $Peak_{i,m}$ representing the m^{th} vector of the element representing the i^{th} randomly generated peak of the multimodality problem. The $\mathbb{P}$ peaks are selected at random from the set of all possible elements X.

To evaluate the performance of a GA with a test-problem generator, Spears begins by randomly generating a set of multiple test problems. The GA is then run once per problem, and the results are averaged across all problems in the set. The monitored performance marker is the best fitness level in the population P at the given time g. This thesis will follow the same approach for evaluating the performance of the CGA.

3.7.2. Empirical and Statistical Results

To see how the different genetic operators effect the performance of the CGA, test cases are run using different combinations of selection, crossover, and mutation. For a given test case, 24 multimodal test problems are generated at random using the test-problem generator. Each test problem has 5 randomly located peaks (global optima). The CGA is run once per problem, and the results are averaged across the 24 problems. In all cases, the population size N is 100 and the number of complex continuous vectors M is 10. The crossover rate $p_c = 0.7$ and the mutation rate $p_m = 0.1$ are set at typical values [25, 14]. The search space S of the test problems is $[-1 < \Re < 1]$ and $[-1 < \Im < 1]$. This space is used for generating the initial population of elements and for randomly selecting the peaks of the test problems. Any vectors that are forced outside the search space S by crossover or mutation are

allowed to exist and are not forced back into S. The mutation range $A = 0.2$ is sized in a typical proportion to S [14]. The stopping criteria is set to $gen\_max = 20,000$ and $fit\_min = 0$. This forces the CGA to run for all 20,000 generations regardless of the generated solutions. Performance is characterized and shown with a curve that plots the fitness of the best element in the population P at time g.

Simulations were performed with MATLAB 6.5.1 concurrently on six 2.4GHz Pentium 4 PCs using different seeds for every trial. Simulation time averaged 11.6 hours per computer. This may seem like a large amount of time for an optimization algorithm. However, the number of operator combinations and test problems per combination required the CGA to be run for its entirety 1,440 times. This means that the average time for the CGA to run once for 20,000 generations is slightly less than 3 minutes.

Twelve fitness plots generated by the simulation are included in Appendix A. A separate plot is drawn for each selection-mutation combination with multiple curves for the different crossover techniques. This makes it easier to see the effect of crossover, especially normal crossover, on the optimization rate of the CGA. The average minimum fitness and standard deviation produced in the simulation for each combination of selection, crossover, and mutation are also listed in Appendix A.

Statistical analysis is necessary to fully understand the data produced by the CGA simulation. A three-way factorial ANOVA with interaction and a 0.05 significance level was performed with the statistical program SAS 8.2. The goal is to determine if the minimum fitness achieved by the different combinations of operators are significantly different. The output confirmed that both the main effects and interaction effects are statistically significant, as seen in Table 1. Therefore, the choice of selection, crossover, and mutation are indeed important to the optimization abilities of the CGA.

Table 1. Factorial ANOVA results for the CGA

Source	**F Value**	**Pr > F**
SEL	9.95	<0.0001
CROSS	1,821.52	<0.0001
SEL∗CROSS	9.91	<0.0001
MUT	18,174.80	<0.0001
SEL∗MUT	6.93	<0.0001
CROSS∗MUT	1,855.53	<0.0001
SEL∗CROSS∗MUT	10.32	<0.0001

Next, a least significant difference (LSD) test at the 0.05 significance level was performed with SAS 8.2 to identify the main effects. The object of this test is to group the treatments so that the difference in means within a group is less than the LSD. The LSD values calculated for selection, crossover, and mutation are 2.8e-3, 3.6e-3, and 3.2e-3, respectively. The following tables show the LSD results for the three operators. The *t Grouping* column in the tables identifies the group to which a given treatment belongs. Different treatments for a genetic operator that belong to the same group are not significantly different based on the LSD test. The groupings shown do not spread across multiple operators. In other words, group A in the selection LSD test is different than group A in the crossover LSD test.

Table 2 indicates that rank-based selection is significantly better than proportionate selection. It is a little suprising, however, that proportionate selection is not significantly better than no selection at all. Table 3 shows that all crossover methods are significantly different with normal crossover, the new hybrid approach, performing the best. No crossover proved to be the worst approach as expected, but discrete crossover performed much better than originally thought. The results for mutation in Table 4 are a little less enlightening. Basically, the LSD test results show

that mutation is better than no mutation, but the method of mutation implemented does not produce significantly different results.

Table 2. LSD test results for selection

Selection	**Mean**	**t Grouping**
None	8.210e-2	A
Proportionate	7.950e-2	A
Rank-Based	7.577e-2	B

Table 3. LSD test results for crossover

Crossover	**Mean**	**t Grouping**
None	1.474e-1	A
Average	1.105e-1	B
Constant	8.256e-2	C
Discrete	5.376e-2	D
Normal	1.370e-3	E

Table 4. LSD test results for mutation

Mutation	**Mean**	**t Grouping**
None	3.146e-1	A
Uniform	1.154e-3	B
Normal	6.52e-4	B
BGA	6.9e-5	B

Based on the LSD output of main effects, it would appear that the combination of rank-based selection, normal crossover, and any form of mutation would produce the best optimization results for the CGA. This assumption is wrong, however. This

is because both the main and interaction effects must be analyzed before making a conclusion. Consider Figures 31, 32, 33, and 34 in Appendix A. In all four plots, rank-based selection paired with normal crossover produces the best results despite the method of mutation. More importantly, the same pair combined with no mutation has the best results by far even though any other combination of selection and crossover with no mutation produces terrible results. This is an example of an interaction effect. Although using mutation is generally better than using no mutation, an effect between normal crossover and no mutation manifests itself as greatly improved performance. A possible explanation for this is that the mutation operator destroys or overly mutates the offspring vectors already generated by normal crossover.

The simulation and statistical results for comparing different selection, crossover, and mutation strategies for the CGA have led to the conclusion that the best configuration is to use rank-based selection, normal crossover, and no mutation. The algorithm block diagram in Figure 6 paired with these decisions represents the selectively improved CGA form that is used in the remainder of this thesis.

3.8. CGA-Based Filter Design

The development of the CGA provides a means for encoding continuous parameters in a GA. Furthermore, empirical and statistical analysis has identified a preferred set of genetic operators for dealing with the continuous parameters. This work solves some problems associated with GA-based filter design, but still more exist. For example,

- How can a digital IIR filter be efficiently represented in an element?
- How should the fitness of a digital IIR filter be calculated to convey the quality of its magnitude response?
- How can an initial population of digital IIR filters be randomly generated to adequately represent the entire solution space S?

- How should the population be managed to ensure that all filters under consideration are realizable?

CHAPTER 4. FILTER DESIGN ALGORITHM

4.1. Introduction

While the CGA is a generic optimization algorithm, it still needs to be modified to suit specific applications, such as digital IIR filter design. For this reason, a customized application layer has been developed for the CGA to handle the specific format and properties of the filter design problem. These requirements include a method for mapping a filter to an element, evaluating the fitness of a filter, creating an initial population of filters, and ensuring that all filters are realizable.

The final filter design algorithm (FDA) that is encompassed by the CGA modifications is evaluated empirically with a number of filter design problems. First, digital IIR filters are designed and optimized to magnitude responses with known transfer functions such as a Butterworth bandpass and a Chebyshev lowpass filter. This helps to characterize the FDA by examining its optimization abilities and performance. Finally, the FDA is used to design a digital IIR filter with an arbitrary magnitude response that has a transfer function which is relatively hard to determine with traditional methods.

4.2. IIR Filter Properties

Let us begin by defining the transfer function $H(z)$ for a digital IIR filter as [27]

$$H(z) = \frac{N(z)}{D(z)} = \frac{\sum_{i=0}^{\alpha} c_i z^{-i}}{1 + \sum_{i=1}^{\alpha} b_i z^{-i}} = K \times \frac{\prod_{i=1}^{\alpha} (z - z_i)}{\prod_{i=1}^{\alpha} (z - p_i)}. \tag{53}$$

The coefficients of the polynomial form are b_i and c_i. The zeros and poles of the factored form are z_i and p_i, respectively. The gain factor K is necessary for equivalence between the polynomial and factored forms. The order of $H(z)$ is determined by α.

Not all filters defined by $H(z)$ are feasible or implementable. Two properties of

$H(z)$ of concern to this thesis are [1]

- A causal, linear, time invariant (LTI) system with system function $H(z)$ is bounded input bounded output (BIBO) stable if and only if all the poles of $H(z)$ lie inside the unit circle. ($|p_i| < 1$)
- A causal, stable, LTI system with system function $H(z)$ is real if and only if all complex poles and zeros of $H(z)$ have complex conjugate pairs or exist singularly on the real axis.

These properties should be enforced during the design and optimization process of an IIR filter.

4.3. CGA Modifications

Several modifications must be made to the CGA to apply it to the FDA. Methods are needed for mapping filter transfer functions into and out of the GA, controlling and monitoring filters to maintain necessary filter properties, and evaluating the fitness of resulting filter designs.

4.3.1. Filter Mapping

For the CGA to evolve and optimize digital IIR filters, a method for mapping a filter transfer function $H_n(z)$ to an element x_n is needed. Two straightforward methods for doing this include mapping either the coefficients of the polynomial form of $H_n(z)$ or the roots and gain of the factored form of $H_n(z)$ to the vectors of x_n. While both of these options are mathematically equivalent, polynomial coefficients b_i and c_i can have several orders of magnitude of dynamic range necessitating a very large search space S. However, this is not the case for the factored version, thereby prompting its selection. Filter stability requires that all poles p_i of $H_n(z)$ are inside the unit circle, thus, limiting the search space S for p_i. Placement of the zeros z_i, although not dictated by stability, can be restricted to the same region by imposing a minimum-phase requirement. Minimum-phase is accomplished by having all poles

and zeros of $H_n(z)$ inside the unit circle. This has the drawback of greatly restricting the possible phase response that can be achieved by the optimization algorithm, but this does not matter here since magnitude response is the only factor of optimization considered in this thesis.

Using the factored form of $H_n(z)$ shown in (53) and ignoring the gain K_n, $M = 2\alpha$ complex vectors are necessary to map $H_n(z)$ to x_n. Representation in this fashion, however, poses a few problems. For $H_n(z)$ to be real, every pole and zero must have a complex conjugate pair or exist singularly on the real axis. Therefore, for every complex vector $a_{n,m}$ in x_n, there must exist another complex vector $a_{n,k}$ where

$$a_{n,k} = a_{n,m}^*. \tag{54}$$

This relationship between vectors changes the way crossover and mutation can operate. For instance, if crossover generates an offspring with a complex vector $a_{n,m}$, the crossover operator must ensure that a complex vector $a_{n,k}$ that satisfies (54) is also generated. Furthermore, if mutation modifies $a_{n,m}$ by λ, then mutation must also modify $a_{n,k}$ by λ^*.

Removing these vector dependencies allows previously discussed crossover and mutation strategies for the CGA to be used. This is done by removing the vector $a_{n,k}$ from x_n and interpreting every complex vector $a_{n,m}$ as two vectors $a_{n,m}$ and $a_{n,m}^*$. This simplification act does not come without consequence. It requires the number of poles and the number of zeros of $H_n(z)$ to be even, and poles and zeros are no longer able to exist singularly on the real axis. However, the reduction in the size of M and the ability to use regular crossover and mutation operators serves as sufficient justification. Therefore, $\frac{\alpha}{2}$ vectors are needed to represent the zeros and $\frac{\alpha}{2}$ vectors are needed to represent the poles, meaning that $M = \alpha$ total vectors are required and α must be even.

The gain K_n of the transfer function $H_n(z)$ is not mapped into x_n. The reason for this is simple. Whereas pole and zeros locations are restricted to inside the unit circle, the gain factor has a possible range of $0 < K < \infty$. This makes it difficult, if not impossible, to choose reasonable values for K in the CGA. Instead, K_n is not treated as an optimizable parameter but rather calculated directly to minimize the fitness of x_n for a given set of poles and zeros. This will be described in greater detail shortly.

4.3.2. Population Management

Filter design with the CGA requires proper population management for good performance. These tasks include the random generation of the initial population $P(0)$ of filters and the monitoring of vector manipulation within $P(g)$ for all g to keep P within the range of S.

When creating $P(0)$, the user must select the desired order α of $H_n(z)$. A higher order filter is more capable of approximating a desired magnitude response, but increasing α also increases M, along with the complexity and optimization time of the CGA. Therefore, it is usually wise to choose a small α that can still produce acceptable results. Remember that α must be even to conform with the filter mapping strategy used in this thesis.

Once the filter order α is chosen, α zero and α pole locations must be randomly selected for each x_n in $P(0)$. For real, stable, minimum-phase digital IIR filters, this is quite easy. Zeros and poles must be inside the unit circle and exist in complex conjugate pairs. In compliance with the proposed filter mapping strategy, this is achieved by selecting complex vector values for each conjugate pair with the uniform

joint PDF

$$a_{n,m}^{0} = Re^{j\Theta} \tag{55}$$

$$f_{R,\Theta}(r,\theta) = \begin{cases} \frac{1}{2\pi r^2}, & 0 \leq r \leq 1,\ 0 \leq \theta < \pi \\ 0, & \text{otherwise} \end{cases} . \tag{56}$$

Generating filters that reside in S is one thing. Keeping filters of subsequent generations within S is another. The filter mapping strategy prevents crossover and mutation from generating and modifying offspring elements that do not conform to the complex conjugate pole/zero pair requirement, but a method for preventing or recovering from pole and zero placement outside the unit circle is needed. The strategy proposed in this thesis is to allow crossover and mutation to manipulate vectors without regard to S. Then, before the replacement step of the CGA, any vectors representing poles and zeros outside the unit circle are mapped back into the unit circle.

A zero mapping function that does not effect the shape of the filter magnitude response is chosen. Any zero z_i that lies outside the unit circle is mapped to a zero $z_i^{'}$ inside the unit circle with the equation

$$z_i^{'} = \frac{1}{z_i^{*}}. \tag{57}$$

This transformation effects the inherent gain of $H_n(z)$, but this is acceptable since K_n is calculated afterwards to minimize fitness.

The shape of the filter magnitude response cannot be preserved when mapping poles from outside to inside the unit circle. Therefore, the proposed pole mapping strategy was developed based on different criteria. Assuming that all poles are located within the unit circle before crossover and mutation are applied, the distance a given

offspring pole is pushed outside the unit circle, if at all, is based on the search ability of the operators. A large, course search factor is needed to push offspring poles a relatively long distance outside the unit circle. Likewise, offspring poles a relatively short distance outside the unit circle can be potentially generated by a small, fine search factor. The pole mapping strategy needed to maintain stability should operate in a similar fashion. Surprisingly enough, this is seen in (57). The reciprocal nature of the equation maps poles close to the unit circle to another location close to the unit circle and poles far from the unit circle to a location closer to the origin. Therefore for sake of simplicity, (57) with z_i replaced by p_i when necessary is used to correct both violating zeros and poles in the FDA.

Note: Complex conjugation of z_i in (57) is not necessary in the FDA since poles and zeros are already required to exist in complex conjugate pairs. It is written this way, however, for applicability to the general case.

4.3.3. Filter Fitness

The goal of the FDA is to design and optimize a digital IIR filter with an arbitrary magnitude response. Thus, the fitness function of the CGA should be based on both the magnitude responses of the filter undergoing evaluation and the desired magnitude response. A frequency weighted squared error technique is proposed for this. The fitness of x_n is calculated by first mapping the vectors of x_n to the pole and zero pairs of $H_n(z)$. Next, the magnitude response $\left|H_n(e^{j\Omega})\right|$ of $H_n(z)$ with a default gain of $K = 1$ is evaluated for all frequency bins $\mathbf{\Omega}$. The desired magnitude response $\left|H_d(e^{j\Omega})\right|$ must also be identified at these same frequency bins. To compensate for the use of unity gain in $H_n(z)$, $\left|H_n(e^{j\Omega})\right|$ is scaled by K_n, where K_n is chosen to minimize the error between $K_n\left|H_n(e^{j\Omega})\right|$ and $\left|H_d(e^{j\Omega})\right|$. This is achieved by forcing the average magnitude value of $K_n\left|H_n(e^{j\Omega})\right|$ to equal the average magnitude value

of $\left|H_d(e^{j\Omega})\right|$. The equation for calculating K_n is

$$K_n = \frac{\sum_{y=1}^{Y} \left|H_d(e^{j\Omega_y})\right|}{\sum_{y=1}^{Y} \left|H_n(e^{j\Omega_y})\right|}. \tag{58}$$

Next, the squared error is calculated by squaring the difference between $K_n \left|H_n(e^{j\Omega})\right|$ and $\left|H_d(e^{j\Omega})\right|$ for all $\mathbf{\Omega}$. The squared error values are then weighted by multiplying them with a weighting vector $\mathbf{Q}$ that assigns a weighting factor to each frequency bin $\mathbf{\Omega}$. This enables certain frequency bins of the magnitude response to contribute more or less to the overall fitness of x_n. Finally, the weighted squared error values are summed and scaled to produce the fitness value of x_n. If $K_n \left|H_n(e^{j\Omega})\right|$ is identical to $\left|H_d(e^{j\Omega})\right|$, then the fitness value will be zero. The complete fitness function used by the FDA is

$$f(x_n) = \frac{1}{Y} \sum_{y=1}^{Y} \left[K_n \left|H_n(e^{j\Omega_y})\right| - \left|H_d(e^{j\Omega_y})\right|\right]^2 Q_y, \tag{59}$$

where Y is the total number of frequency bins, Ω_y is an element of $\mathbf{\Omega}$, and Q_y is an element of $\mathbf{Q}$. Since only real filters are considered for $H_n(z)$, $\mathbf{\Omega}$ is generally only specified in the range of 0 to π. Any distribution of frequency points, such as linear or logarithmic spacing, within $\mathbf{\Omega}$ can be applied, with varying results on the outcome of the algorithm.

4.4. FDA Empirical Analysis

The proposed FDA is tested and evaluated with three digital IIR filter design problems where $\left|H_d(e^{j\Omega})\right|$ has a known solution $H_d(z)$. While it should be theoretically possible for the FDA to optimize to the global solution for any desired magnitude response, this setup should force the FDA to design a filter that is exactly identical to $H_d(z)$ provided that an adequate order is specified. Therefore, the

optimized solution should have a fitness value of zero. Any fitness greater than zero represents the effectiveness of the FDA in identifying the exact global solution.

Next, the FDA is applied to a digital IIR filter design problem where $\left|H_d(e^{j\Omega})\right|$ has either an unknown or hard to find transfer function $H_d(z)$. This highlights the abilities of the FDA to design a digital IIR filter transfer function that is difficult with traditional techniques.

4.4.1. Test Case I

The Butterworth filter equation is selected for the first test because it identifies a well-known, mathematically optimized class of filters that are maximally flat in the pass band. The desired magnitude response $\left|H_d(e^{j\Omega})\right|$ is a fourth-order Butterworth bandpass filter with lower and upper 3-dB cutoff points of $\Omega_l = \frac{\pi}{4}$ and $\Omega_u = \frac{3\pi}{4}$, and unity passband gain. The frequency vector $\mathbf{\Omega}$ for specifying $\left|H_d(e^{j\Omega})\right|$ and evaluating $K_n \left|H_n(e^{j\Omega})\right|$ consists of 10,000 frequency bins equally spaced between 0 and π. The weighting vector $\mathbf{Q}$ equals 1 for all 10,000 points. This forces all frequency bins of the magnitude response to be equally important for optimization.

The FDA is configured for the test to accommodate the exact Butterworth transfer function output. This is accomplished by setting α equal to four, thereby directing the FDA to design a filter that is the same order as $H_d(z)$. The other parameters of the FDA are intelligently chosen from experience gained from CGA testing. They are as follows: population size $N = 200$, element size $M = \alpha = 4$, and probability of crossover $p_c = 0.7$. The exit criteria are set to $gen\_max = 2,000$ and $fit\_min = 0$ so that the FDA will search for the optimal solution for 2,000 generations.

The magnitude response of the designed filter is shown in Figure 7, and the corresponding pole-zero plot is shown in Figure 8. For comparison, the pole-zero plot for the exact fourth order Butterworth bandpass filter is included in Figure 9. The

errors between the pole-zero placements in the desired and designed filters are listed in Table 5. The poles, zeros, and gain factor for the desired Butterworth filter were generated automatically by the MATLAB Signal Processing Toolbox.

The fitness curve in Figure 10 shows that the ending fitness level of the designed filter is approximately 3e-32 and that major fitness improvements ceased after about 1,300 generations. The FDA could have been stopped before executing the last 700 generations without much impact on results, but neither of the two stopping criteria were met to cause the algorithm to cease. The addition of new stopping criteria related to fitness rate of change could have been incorporated to cause the FDA to halt after the first 1,300 generations in this case, but this is beyond the scope of this thesis.

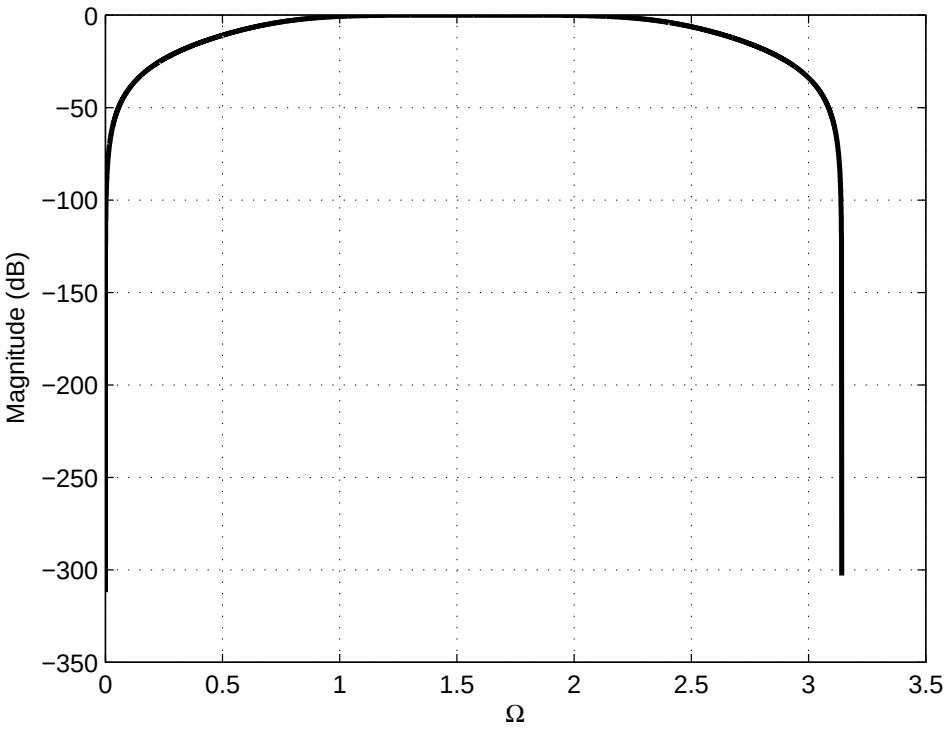

Figure 7. Magnitude response of the designed filter (I).

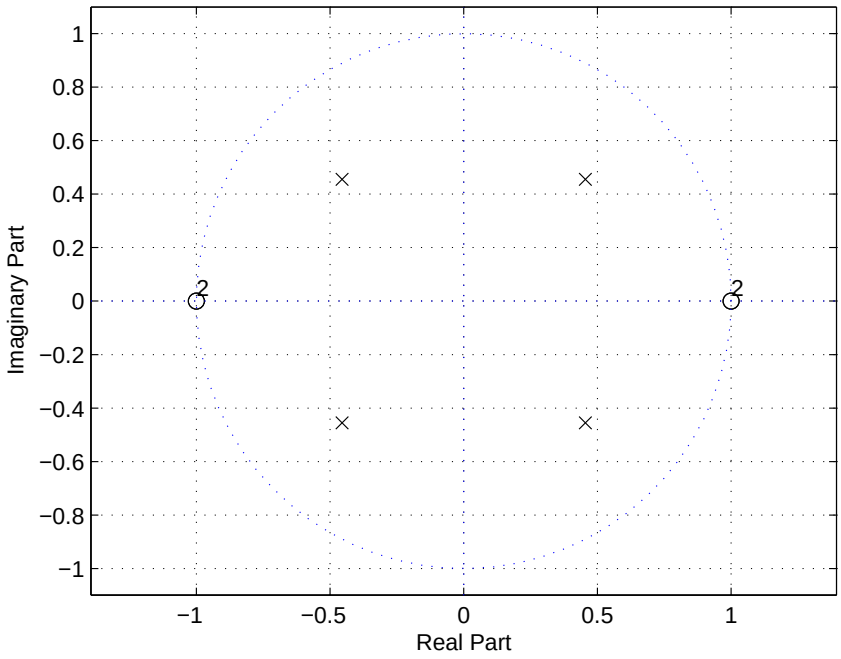

Figure 8. Pole-zero plot for the designed filter (I).

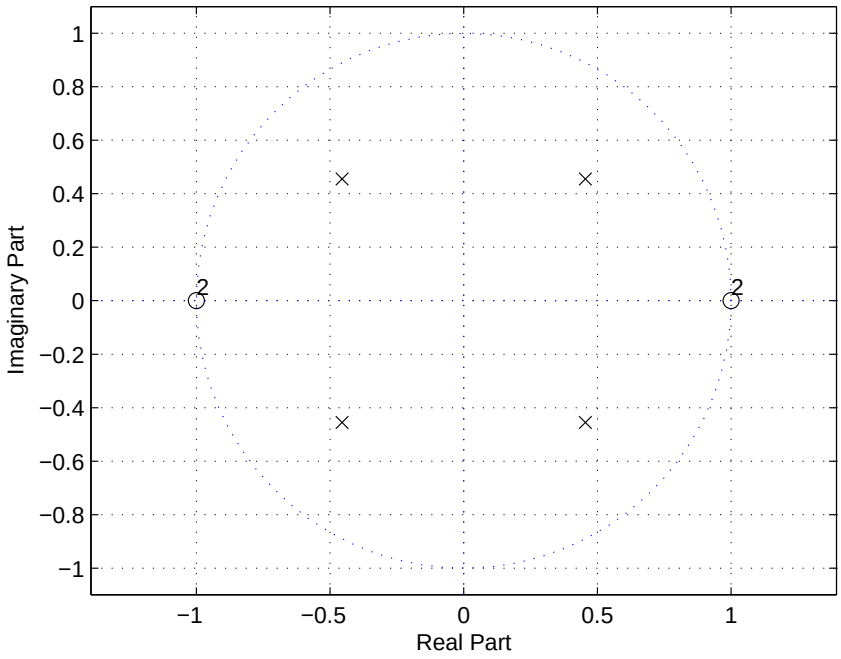

Figure 9. Pole-zero plot for the desired filter (I).

Table 5. Results for test case I

Parameter	**Desired**	**Designed**	**Error** [%]
Gain	0.29289	0.29289	2.4602e-6
Zeros	1±j0	1.0000±j9.5758e-9	1.2769e-6
	-1±j0	-1.0000±j2.0142e-8	2.5820e-6
Poles	0.45509±j0.45509	0.45509±j0.45509	3.4501e-14
	-0.45509±j0.45509	-0.45509±j0.45509	4.3126e-14

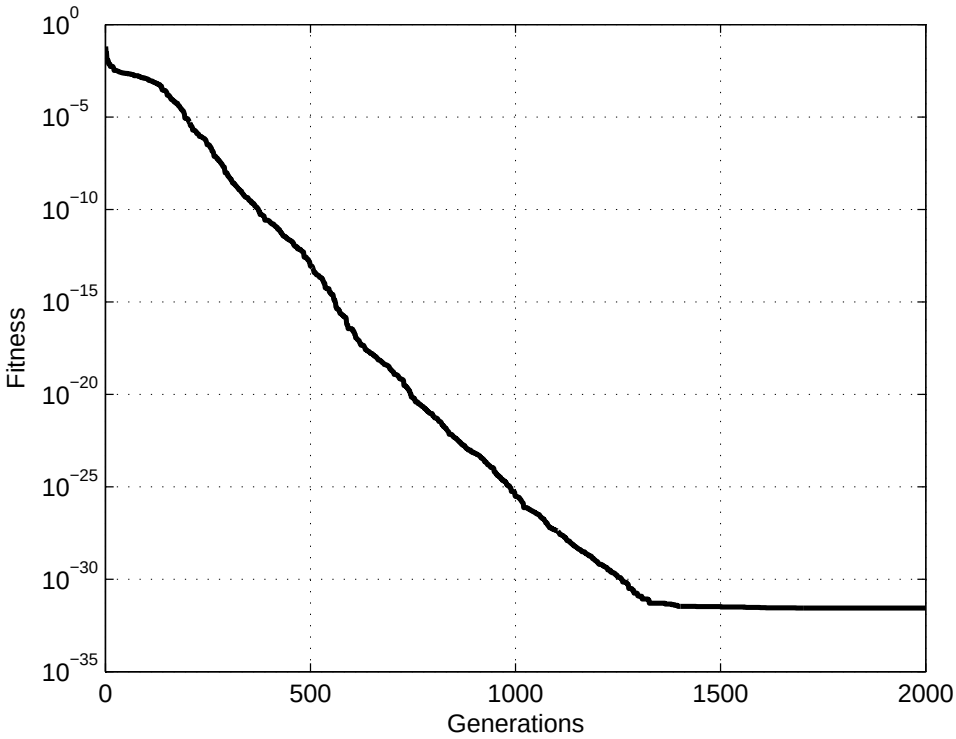

Figure 10. Fitness curve of the FDA (I).

4.4.2. Test Case II

The second FDA test is to design a Chebyshev filter. The Chebyshev filter equation represents another mathematically unique class of filters that achieve sharper transition bands by incorporating ripple into the pass band. The desired magnitude

response $\left|H_d(e^{j\Omega})\right|$ is a fourth-order Chebyshev lowpass filter with a 3-dB cutoff point of $\Omega_c = \frac{\pi}{2}$, unity passband gain, and passband ripple of 0.5 dB. The frequency vector $\mathbf{\Omega}$, weighting vector $\mathbf{Q}$, and all other configuration parameters for the FDA are set exactly the same as in test case I. Therefore, the only difference in this test is the desired magnitude response input.

The magnitude response comparison plot in Figure 11 shows that the FDA output closely matches the desired filter magnitude response to about 100 dB down. Pole-zero plots for the designed and desired filters are shown in Figures 12 and 13.

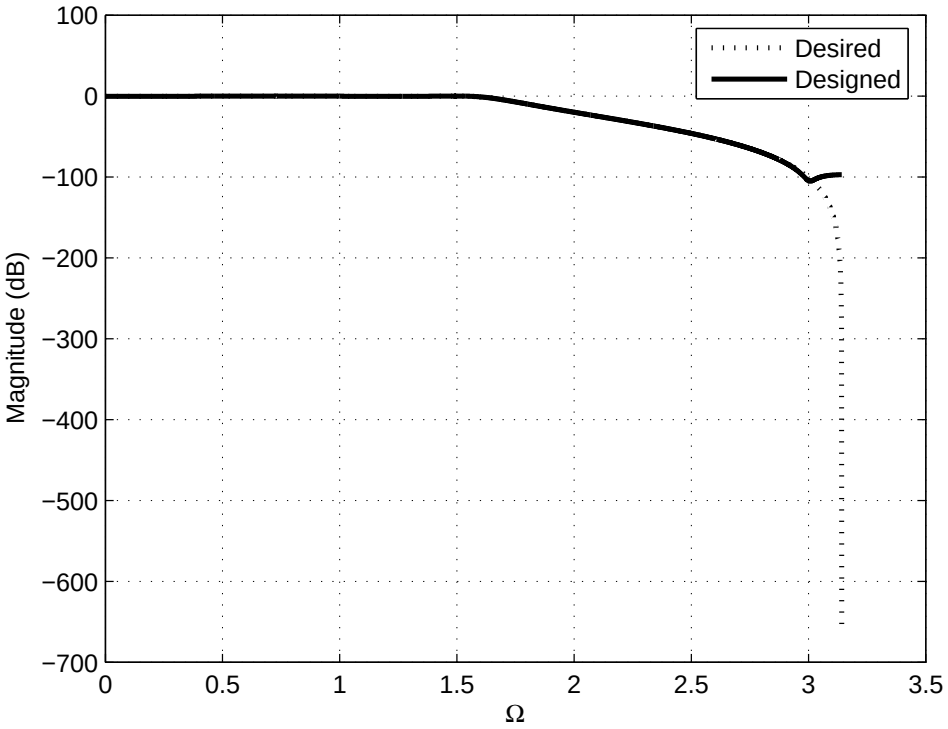

Figure 11. Magnitude response comparison (II).

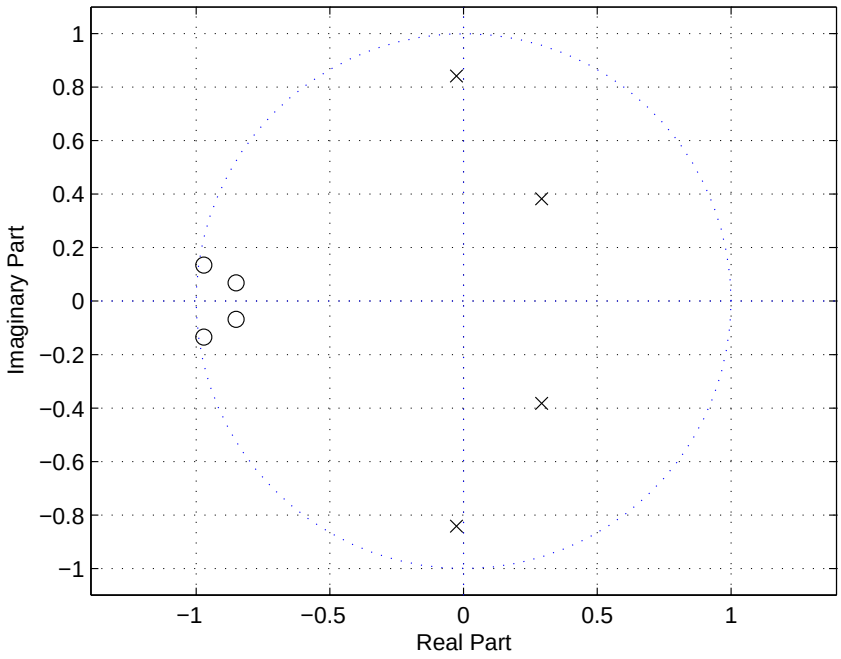

Figure 12. Pole-zero plot for the designed filter (II).

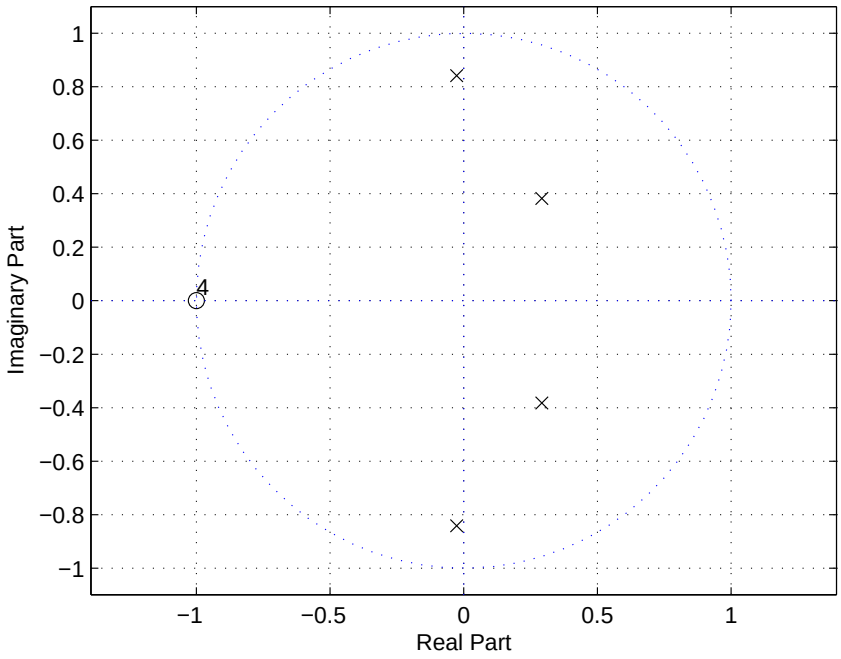

Figure 13. Pole-zero plot for the desired filter (II).

The percent errors between the pole and zero locations and gain factors for the desired Chebyshev lowpass filter and the FDA-designed filter are listed in Table 6. As in the Butterworth case, the desired lowpass Chebyshev filter transfer function was designed with the MATLAB Signal Processing Toolbox. Figure 14 shows that the minimum fitness achieved was approximately 6e-12 and that convergence was achieved after about 1000 generations.

Table 6. Results for test case II

Parameter	**Desired**	**Designed**	**Error** [%]
Gain	6.7280e-2	8.0349e-2	19.4250
Zeros	-1±j0	-0.8510±j6.7966e-2	16.3754
	-1±j0	-0.9715±j0.1347	13.7715
Poles	0.2921±j0.3821	0.2921±j0.3821	4.2796e-4
	-2.6310e-2±j0.8419	-4.5509e-1±j4.5509e-1	5.3867e-5

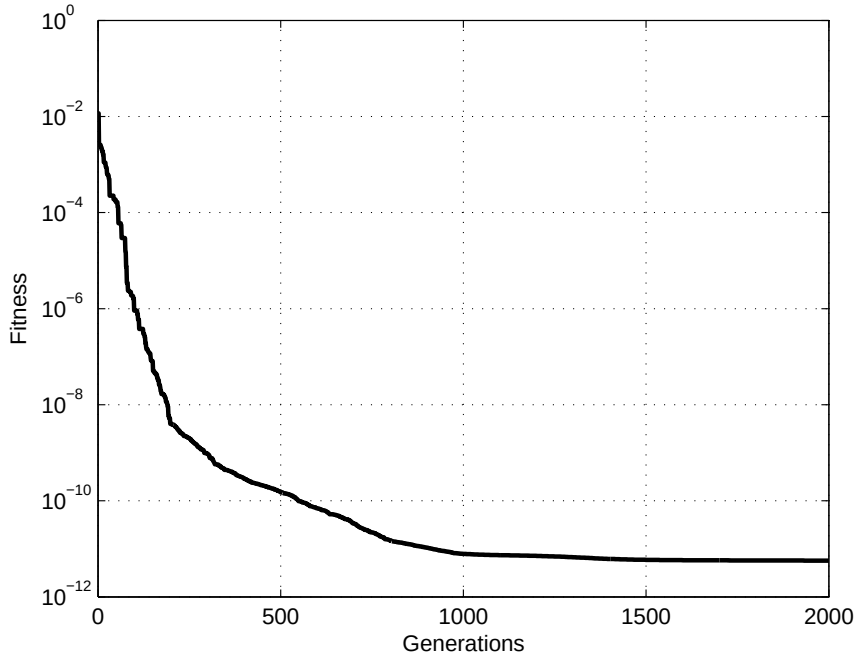

Figure 14. Fitness curve of the FDA (II).

4.4.3. Test Case III

Test case III is identical to the Butterworth filter in test case I except that a sixth-order filter is being designed to fit a desired fourth-order filter response. The goal of this test is to see if the FDA can optimize a filter that is of higher order than necessary for the desired magnitude response. The desired outcome of this test would be a fourth-order Butterworth bandpass filter transfer function with two extra poles and zeros that cancel, essentially rendering a fourth-order filter.

The pole-zero plot of the FDA-designed filter is shown in Figure 15. Comparing this to the ideal fourth-order Butterworth bandpass pole-zero plot in Figure 9 shows that, as expected, the extra poles and zeros cancel out, leaving behind the poles and zeros necessary for the desired fourth-order filter.

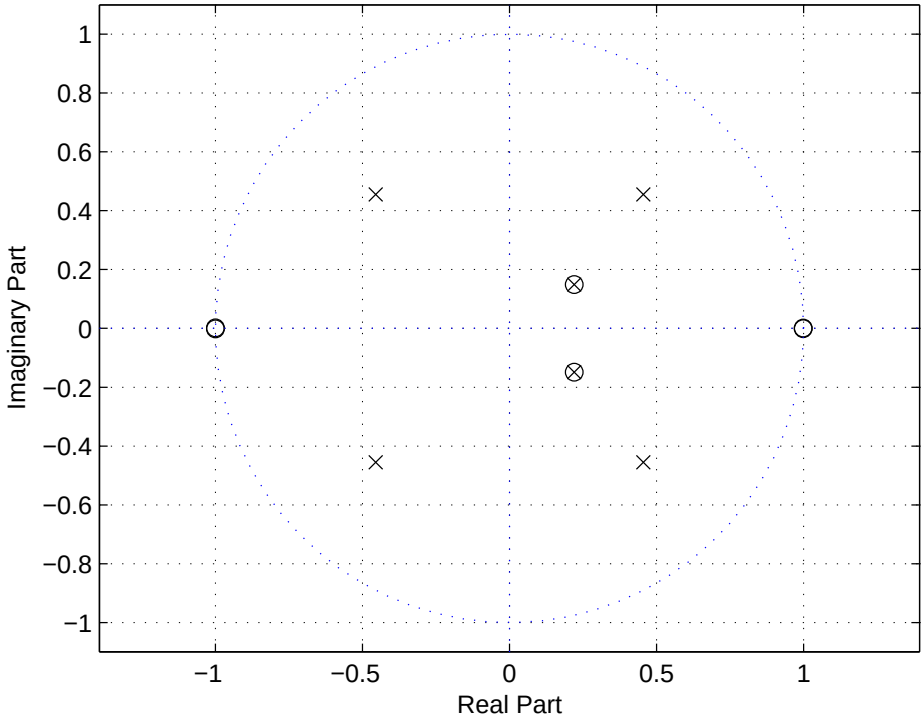

Figure 15. Pole-zero plot for the designed filter (III).

4.4.4. Test Case IV

Developing a new technique for designing Butterworth and Chebyshev filters is no great accomplishment, especially when compact mathematical equations already exist. However, the new technique would be very useful if it can also design filters which do not have such equations. To demonstrate the FDA's ability to do just that, the algorithm is tested with the task of designing a transfer function for a slightly more complicated magnitude response.

The goal of this test case is to design a digital IIR filter transfer function with a magnitude response resembling a lowpass filter superimposed with two notch filters in the passband. The specifications for the desired digital IIR filter are as follows:

- sampling rate of $\Omega_s = 1000Hz$
- ideal lowpass cutoff at $\Omega_c = 300Hz$
- first ideal notch filter at $\Omega_{n1} = 60Hz$
- second ideal notch filter at $\Omega_{n2} = 120Hz$
- passband gain of one
- tenth-order transfer function

A filter with these characteristics would be useful for removing the high frequency content of a signal along with 60 and 120 Hz noise possibly injected by AC lines and fluorescent lights.

The desired magnitude response that is given as an input to the FDA is shown in Figure 16. The desired magnitude response has a passband with a gain of one, a notch at 60 Hz with a bandwidth of 12 Hz, a notch at 120 Hz with a bandwidth of 24 Hz, and a lowpass cutoff point at 300 Hz. The desired response is specified for 10,000 logarithmically spaced points between 0.5 Hz and 500 Hz. This type of spacing

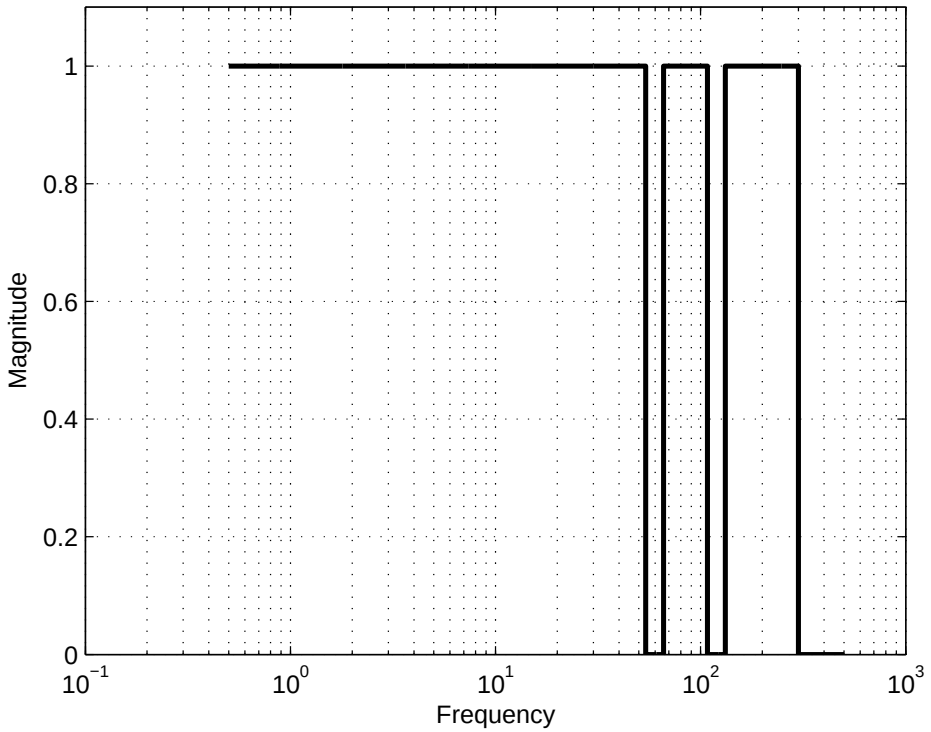

Figure 16. Desired magnitude response (IV).

combined with the 20% notch bandwidths gives equal weighting to both notches by specifying the same number of frequency bins in each. When transformed into the digital domain, the frequency vector $\mathbf{\Omega}$ specifies $\left|H_d(e^{j\Omega})\right|$ from 0.001π to π. The weighting vector $\mathbf{Q}$ is set to 1 for all 10,000 points, except for within the two notches, where $\mathbf{Q}$ equals five. This makes the FDA match the frequency bins in the notches five times better than the rest of the frequency bins for the desired magnitude response. Additional FDA settings are as follows: $N = 500$, $M = \alpha = 10$, and $p_c = 0.7$. The exit criteria are set to $gen\_max = 2,000$ and $fit\_min = 0$ so that the FDA will search for the optimal solution for 2,000 generations.

The magnitude response of the tenth order filter designed by the FDA after the 2,000 iterations is shown in Figure 17, and the corresponding pole-zero plot is

shown in Figure 18. Looking at Figure 19 shows that the majority of the FDA design took only 1,100 generations to complete. The designed filter closely meets all of the specified requirements of the desired filter. The percent error between the cutoff points and DC gain for the desired and designed filters are listed in Table 7. The resulting 3-dB bandwidths for the 60 and 120 Hz notch filters are 22.73 and 42.72 Hz, respectively. The attenuation achieved at 60 Hz is -36.4 dB, and 120 Hz attenuation is -41.06 dB. The stopband ripple of the lowpass filter is -23.5 dB, and the maximum passband ripple is 1.22 dB.

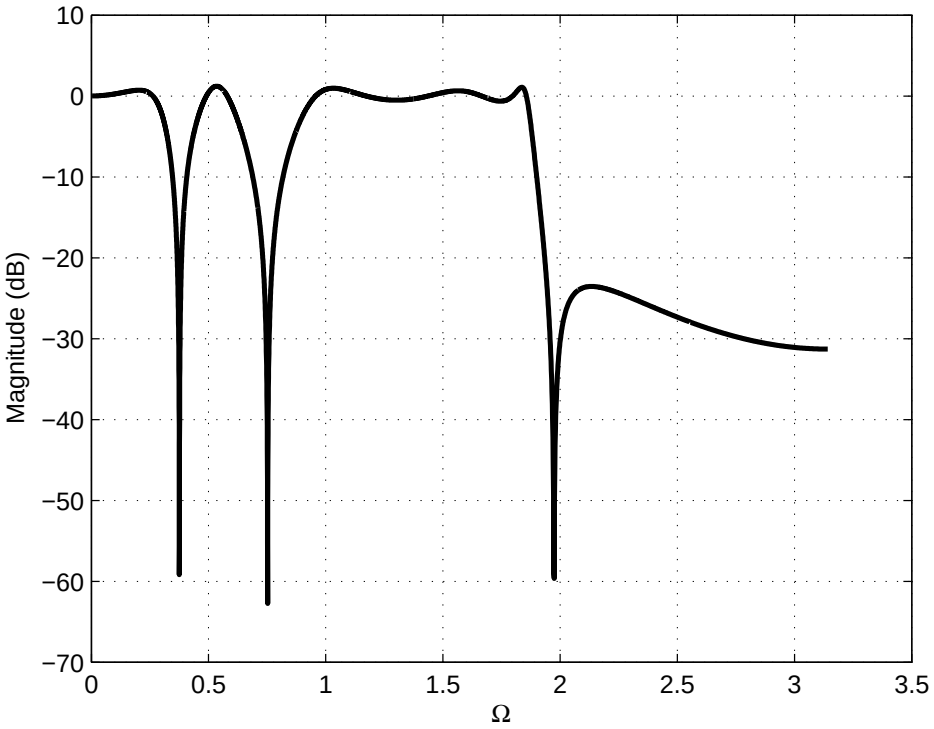

Figure 17. Magnitude response of the designed filter (IV).

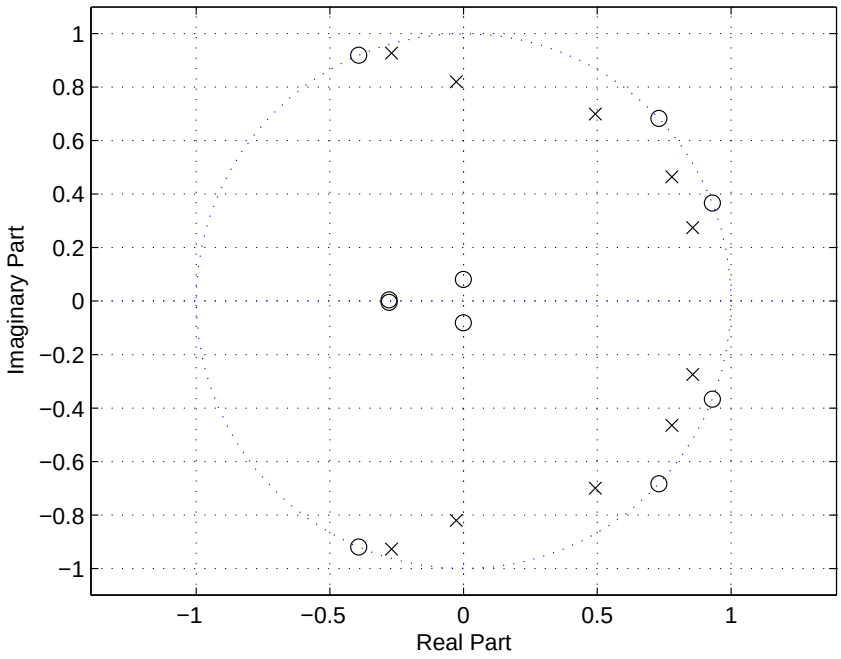

Figure 18. Pole-zero plot for the designed filter (IV).

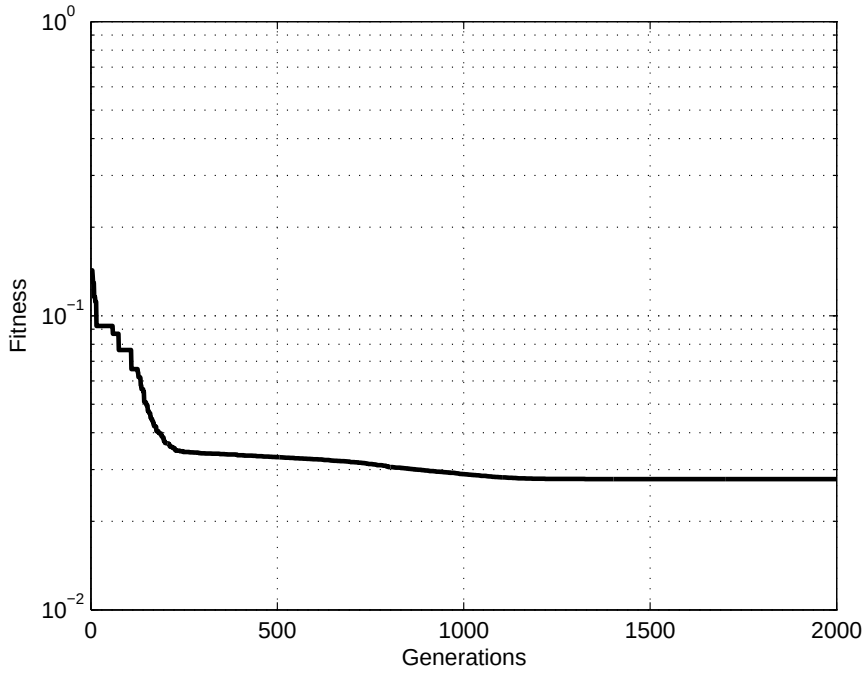

Figure 19. Fitness curve of the FDA (IV).

Table 7. Results for test case IV

Parameter	**Desired**	**Designed**	**Error** [%]
Lowpass 3-dB cutoff Ω_c	300 Hz	298.1 Hz	-0.63
First Notch Ω_{n1}	60 Hz	59.76 Hz	-0.40
Second Notch Ω_{n2}	120 Hz	119.76 Hz	-0.20
DC Gain	1.0	1.0006	0.06

4.4.5. Test Case V

Test case V is identical to test case IV except that the logarithmic spacing used for $\mathbf{\Omega}$ is replaced with a linear spacing scheme. The goal of this test is to see the effect of different frequency point distributions on the FDA output. The magnitude response of the linear and logarithmic spaced filters are compared in Figure 20.

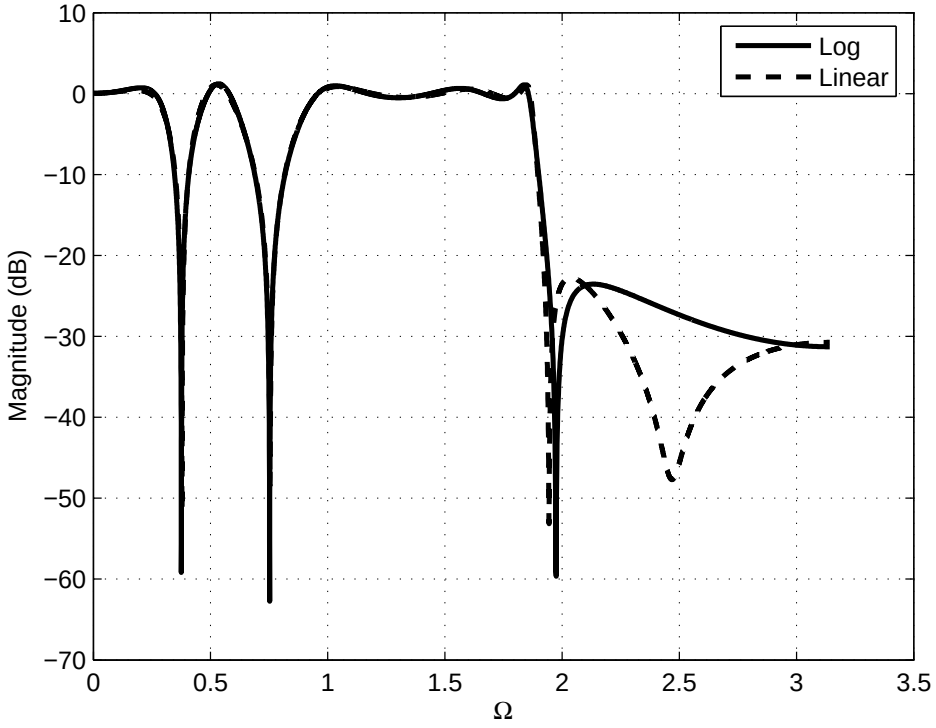

Figure 20. Test case IV and V filter comparison.

The linearly spaced designed filter closely meets all of the specified requirements of the desired filter. The percent error between the cutoff points and DC gain for the desired and designed filters are listed in Table 8. The resulting 3-dB bandwidths for the 60 and 120 Hz notch filters are 22.03 and 42.93 Hz, respectively. The attenuation achieved at 60 Hz is -37.9 dB, and 120 Hz attenuation is -47.65 dB. The stopband ripple of the lowpass filter is -22.76 dB, and the maximum passband ripple is 1.11 dB.

Table 8. Results for test case V

Parameter	**Desired**	**Designed**	**Error** [%]
Lowpass 3-dB cutoff Ω_c	300 Hz	298.61 Hz	-0.46
First Notch Ω_{n1}	60 Hz	60.16 Hz	0.27
Second Notch Ω_{n2}	120 Hz	119.96 Hz	-0.03
DC Gain	1.0	1.0077	0.77

CHAPTER 5. DISCUSSIONS

The results for the five test cases presented in the previous chapter demonstrate that the CGA-based FDA is capable of designing a digital IIR filter approximating a desired magnitude response. The fourth-order Butterworth bandpass filter designed in test case I almost exactly matches the desired response. The fitness level of the filter is 3e-32 and could very well be an artifact of round-off error. Obtaining results much better than this could be difficult considering the discrete processing nature of the algorithm in Matlab. Test case I shows that the FDA can indeed find the transfer function required for a given magnitude response. This is apparent in the small errors between the actual Butterworth pole-zero locations and the FDA-designed pole-zero locations.

Test case II helps to confirm the results seen in test case I. The fourth-order filter designed by the FDA closely matches the desired fourth-order Chebyshev lowpass filter. While the fitness level was not quite as good as in test case I, the resulting magnitude response tracked the actual Chebyshev response to around -100 dB. The percent errors for the gain factor and zero locations are fairly high, but the optimization of pole locations was quite good. Pole placement in the Butterworth case was also better than zero location and gain factor optimization. This may indicate that pole placement has a larger effect on the magnitude response of the filter and, thus, has a larger influence on the fitness level of the filter. This would force the FDA to more heavily concentrate on the pole locations to reduce fitness. This is only a hypothesis and is in no way meant to be a generalization of FDA performance and ability.

The CGA inside the FDA only operates on the pole and zero locations of the filters. The necessary gain factors are calculated directly in order to minimize the fitness. Therefore, the large percent error for the gain factor in the Chebyschev case

is an indication of the FDA attempting to compensate for the relatively poor zero location optimization.

The purpose of test case III is to see if the FDA is still capable of designing a filter if the specified filter order for the FDA is higher than necessary. This case adds a new twist to the CGA because an infinite number of solutions to the problem exist. The resulting sixth-order filter designed in the test case, however, had no problem achieving the desired fourth-order Butterworth bandpass filter response. The extra poles and zeros are placed at the same location and canceled by the FDA, thus reducing the problem to a fourth-order filter.

Test case IV attempts to design a filter with a more complicated magnitude response. A transfer function does not exist for the desired ideal filter response so the FDA is used to approximate a solution with a tenth-order filter. A tenth-order filter was selected for the design because it should be of high enough order to achieve two notches and a lowpass shape in the filter response. A higher order could have been selected, and this may well have provided a better solution, but it would also take longer for the FDA to complete the design. The pole-zero plot of the designed filter shows that the FDA did not fully exploit the abilities of the poles and zeros to optimize the magnitude response. Pairs of zeros are seen near the unit circle, where the notch filters and low pass filter cutoffs occur. However, two other pairs of zeros are found near the origin. Placing these closer to the unit circle between the angular frequencies corresponding to 300 and 500 Hz would help the FDA better control the stopband attenuation of the lowpass section of the filter.

The logarithmic spacing used in test case IV helps the FDA better fit the notch filters in the low frequency band of the filter since more frequency points are used in this area. However, this means that relatively few frequency bins are used to calculate the fitness of the filter for higher frequencies. This causes the FDA to concentrate

less on matching the desired response in the high frequency band. This results in poor stopband attenuation. To counter this, test case V is performed with linearly spaced frequency bins to give more weight to the stopband of the filter. The resulting filter of test case V is nearly identical to the filter in test case IV, and the stopband attenuation is actually a little worse in the filter designed with linear spaced frequency bins. Despite this, it can be seen from the magnitude responses that the stopband of the filter in test case V may peak higher but overall is much lower than the filter from test case IV. This shows that the linear spacing does help the FDA better fit the higher frequencies of the filter response.

The input parameters chosen for the FDA have an effect on the design outcome of the algorithm. The main factors include population size N, filter order α, maximum number of generations gen_max, frequency vector $\mathbf{\Omega}$, and weighting vector $\mathbf{Q}$. The complexity of the desired magnitude response plays a role in determining the appropriate value for some of these factors. For example, test cases I, II, and III have $N = 200$ whereas test cases IV and V have $N = 500$. This was necessary because early tests showed that filters of higher orders, thus more vectors, require a larger population to have an adequate variation of filters for the CGA to process.

Likewise, more complex magnitude responses require the design of higher order filters. The exact choice of order depends on the quality of solution desired. The frequency vector $\mathbf{\Omega}$ and weighting function $\mathbf{Q}$ depend on the type of filter being designed and the quality of solution desired. Generally, linear spaced frequency bins should suffice for most problems. In test cases IV and V, initial testing showed that a higher weighting value is needed for the notch filters to force the FDA to emphasis these points more. When uniform weighting was used, the FDA tended to ignore the notch filters and accept the minor fitness penalty from not matching these points. Adding extra frequency bins to certain areas of the response can also force the FDA to

better fit these sections. Selection of parameters is often a guess and check procedure and may require multiple designs to be evaluated by the FDA before desired results are seen.

CHAPTER 6. FUTURE WORK AND CONCLUSIONS

Several aspects of the CGA-based FDA can use improvement or additional analysis. Do the statistical results from the CGA operator tests adequately apply to the filter design problem? Additional testing of the FDA with different genetic operators would be useful in determining if the combination of rank-based selection, normal crossover, and no mutation is indeed the best for filter design. This is a difficult problem to attack, however, since there are a wide range of possible magnitude responses to design. Determining a "general" case magnitude response or a sufficient test bed of magnitude responses is not intuitively obvious. It would also be interesting to re-run the CGA operator tests using different parameters. Using a search space of the unit circle and forcing all vectors to remain inside the unit circle throughout the test would better approximate the situation seen in the filter design problem.

Second, the encoding method chosen for the CGA may not be optimal for the FDA. Currently, vectors are encoded as double precision floating-point complex numbers in rectangular coordinates. Two consequences of this choice in regard to FDA performance are that cartesian coordinates are not necessarily the best way to identify points in a circular search space and that the range of floating-point numbers means that only a small fraction of possible values are used with a unit circle search space. Switching to polar coordinates would be better for representing poles and zeros very near the unit circle where accuracy is important, and switching to a scaled binary encoding scheme with a full range of the unit circle would help to increase accuracy even more.

Other ideas for improvement include adjusting the amount of mutation in the CGA depending on the location of poles and zeros in the unit circle. This is because the magnitude response is more sensitive to the movement of poles and zeros near the unit circle compared to poles and zeros near the origin. Furthermore, other

forms of normal crossover should be considered, such as using a bivariate distribution rather than a univariate distribution for generating offspring. A bivariate distribution would allow for greater exploration of the search space while still exploiting the parent vectors.

Third, allowing a variable filter order may be beneficial. Instead of specifying the desired filter order, the FDA would automatically select the order of the filter needed to match the desired magnitude response. This could be incorporated as another parameter for optimization in the FDA. Difficulties could arise, however, when determining how to genetically operate on a set of filters of varying order. A possible solution could be to start by designing low order filters and then cascade additional low order sections as the need for a higher order filter is determined.

Additional tests have shown that the FDA in its current form has trouble designing filters of higher orders than those already presented in this paper. For example, one such test case consisted of designing a twentieth-order digital IIR filter to match the magnitude response of a 30 band graphic equalizer configured for Festival Concert Hall at North Dakota State University. The best filter designed by the FDA after 10,000 generations is compared to the actual graphic equalizer response in Figure 21. The minimum fitness curve for the design is shown in Figure 22. The designed magnitude response matches the most basic shape of the graphic equalizer but not the individual peaks and valleys. The fitness curve also shows that very little progress was made in improving the fitness of the designed filter over the 10,000 generations.

Improvements to the FDA are necessary to better handle higher order filters and more complex magnitude responses. Furthermore, running the FDA for 10,000 generations, even for a fairly small population size, can take a number of hours to complete. This is primarily due to the time it takes to calculate the magnitude responses of the sample filters for fitness evaluation. Because of the nature of

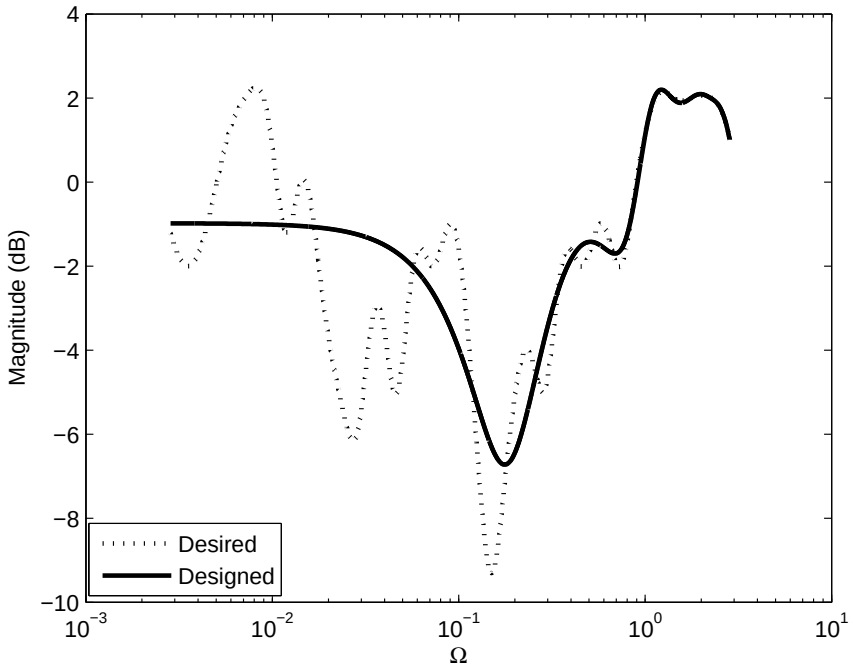

Figure 21. Magnitude response comparison (Festival Equalizer).

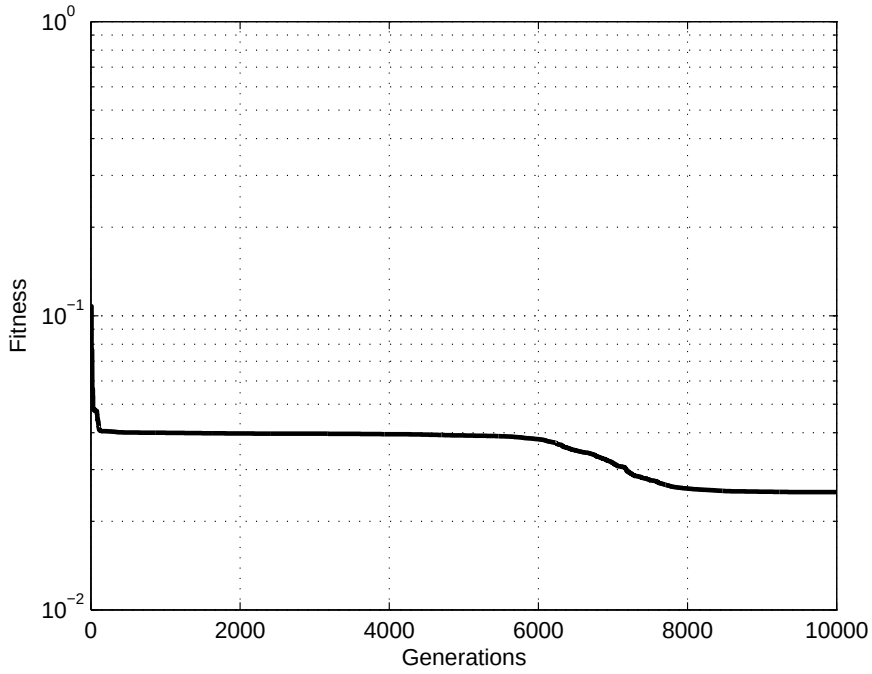

Figure 22. Fitness curve of the FDA (Festival Equalizer).

genetic algorithms, there are often repeat vectors found in the population. It may be possible to exploit this fact by dividing all filters into cascaded second-order sections, evaluating each unique second order filter once, and multiplying the responses to find the overall magnitude responses of the filters. There is a good possibility that there will be multiple instances of the same second-order filters in the population, thus reducing the number of magnitude responses that must be calculated.

Simulations show that is it possible to optimize a system with complex, continuous parameters with a GA. A new genetic operator, normal crossover, has been proposed to better handle crossover and mutation on an adaptive scale to account for system convergence. Statistical analysis of the optimization results for a randomly generated, multimodal objective function shows that improved optimization in a GA can be achieved by replacing the usual crossover and mutation operators with the normal crossover operator.

An FDA built upon a GA using rank-based selection and normal crossover is developed for the design of digital IIR filters with arbitrary magnitude responses. Methods for mapping, monitoring, and evaluating filter transfer functions within the embedded GA are proposed and tested. Test results show that the FDA is able to optimize a digital IIR transfer function that nearly matches a desired fourth-order bandpass Butterworth filter exactly. In a second test, the FDA designed a transfer function the closely approximated a fourth-order Chebyshev lowpass filter. While results were not as good as in the Butterworth test, the resulting magnitude response is still considered adequate for most implementations.

FDA results for higher order filters and more complex magnitude responses are mixed. When the FDA is used to design a tenth-order transfer function with a magnitude response resembling a lowpass filter with two notch filters in the passband, results are satisfactory but not ideal. The desired characteristic shape is achieved, but

transition bands are somewhat sloppy. Examination of the pole and zero locations of the designed transfer function indicates that better results are most likely possible. The last test performed with the FDA was to design a twentieth-order digital IIR filter with a response matching a pre-configured graphic equalizer. The resulting filter is by no means satisfactory. This serves as possible indication of large system optimization problems with the FDA or possibly the results of attempting to fit an inadequately sized transfer function to an overly complex magnitude response.

More work is necessary, especially mathematical analysis, to fully characterize the results of this thesis.

REFERENCES

[1] S. Mitra, *Digital Signal Processing: A Computer-Based Approach*, 2nd ed. Boston: McGraw-Hill Irwin, 2001.

[2] R. Mersereau & M. Smith, *Digital Filtering: A Computer Laboratory Textbook*. John Wiley & Sons, Inc, 1994.

[3] D. Goldberg, *Genetic Algorithm in Search, Optimization, and Machine Learning*. Reading, MA: Addison Wesley Pub. Co., 1989.

[4] J. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The Univeristy of Michigan Press, 1975.

[5] C. Darwin, *The Origin of Species*, ser. The Harvard Classics. New York: P F Collier & Son, 1909, vol. 11.

[6] W. Edmonson et al., "A global least mean square algorithm for adaptive iir filtering," *IEEE Trans. on Circuits and Systems*, vol. 45, no. 3, pp. 379–384, Mar 1998.

[7] D. Talla, S. Rao, & L. John, "An evolutionary computation embedded iir lms algorithm," in *Proceedings of International Conference on Signal Processing Applications and Technology*, Orlando, FL, Nov 1-4, 1999.

[8] L. Wang, W. Li, & D. Zheng, "A class of hybrid strategy for adaptive iir filter design." Shanghai, China: 8th International Conference on Neuaral Information Processing, Nov 14-18, 2001.

[9] A. Košir & J. Tasič, "Genetic algorithm and filtering." Sheffield, UK: First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Sep 14-18, 1995.

[10] D. Dumitrescu et al., *Evolutionary Computation.* Boca Raton, FL: CRC Press, 2000.

[11] L. Davis & M. Steenstrup, "Genetic algorithms and simulated annealing: An overview," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. London: Morgan Kaufmann Pub., 1987, pp. 1–11.

[12] L. Booker, "Improving search in genetic algorithms," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. London: Morgan Kaufmann Pub., 1987, pp. 61–73.

[13] J. Grefenstette, "Rank-based selection," in *Evolutionary Computation I: Basic Algorithms and Operators*, T. Bäck, D. Fogel, and Z. Michalewicz, Ed. Bristol, UK: Institute of Physics Publishing, 2000, vol. 1, pp. 187–194.

[14] H. Mühlenbein & D. Schlierkamp-Voosen, "The science of breeding and its application to the breeder genetic algorithm (bga)," in *Evolutionary Computation*, vol. 1. Cambridge, MA: MIT Press, 1993, pp. 335–360.

[15] ——, "Predictive models for the breeder genetic algorithm: I. continuous parameter optimization," in *Evolutionary Computation*, vol. 1. Cambridge, MA: MIT Press, 1993, pp. 25–50.

[16] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, D. Schaffer, Ed. George Mason University: Morgan Kaufmann Pub., Jun 1989, pp. 2–9.

[17] T. Bäck & D. Fogel, "Mutation operators," in *Evolutionary Computation I: Basic Algorithms and Operators*, T. Bäck, D. Fogel, and Z. Michalewicz, Ed. Bristol, UK: Institute of Physics Publishing, 2000, vol. 1.

[18] R. Craighurst & W. Martin, "Enhancing ga performance through crossover prohibition based on ancestory," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. Eshelman, Ed. University of Pittsburgh: Morgan Kaufmann Pub., Jul 1995, pp. 130–135.

[19] L. Eshelman & D. Schaffer, "Preventing premature convergence in genetic algorithms by preventing incest," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. Belew & L. Booker, Ed. University of California, San Diego: Morgan Kaufmann Pub., Jul 1991, pp. 115–122.

[20] D. Schaffer & L. Eshelman, "On crossover as an evolutionary viable strategy," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. Belew & L. Booker, Ed. University of California, San Diego: Morgan Kaufmann Pub., Jul 1991, pp. 61–68.

[21] D. Goldberg, "Simple genetic algorithms and the minimal, deceptive problem," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. London: Morgan Kaufmann Pub., 1987, pp. 74–88.

[22] J. Antonisse, "A new interpretation of schema notation that overturns the binary encoding constraint," in *Proceedings of the Third International Conference on Genetic Algorithms*, D. Schaffer, Ed. George Mason University: Morgan Kaufmann Pub., Jun 1989, pp. 86–91.

[23] C. Janikow & Z. Michalewicz, "An experimental comparison of binary and floating point representations in genetic algorithms," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. Belew & L. Booker, Ed. University of California, San Diego: Morgan Kaufmann Pub., Jul 1991, pp. 31–36.

[24] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer-Verlag, 1992.

[25] K. D. Jong, "Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, Ann Arbor, MI, 1975.

[26] W. Spears, "The role of mutation and recombination in evolutionary algorithms," Ph.D. dissertation, George Mason University, Fairfax, VA, 1998.

[27] A. Williams & F. Taylor, *Electronic Filter Design Handbook*, 3rd ed. New York: McGraw-Hill, Inc, 1995.

APPENDIX A. CGA SIMULATION RESULTS

Table 9. Minimum fitness for SEL=None and Mut=None (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	5.957e-1	4.343e-2
Discrete	1.923e-1	3.283e-2
Average	4.549e-1	5.124e-2
Constant	3.775e-1	4.237e-2
Normal	0.0	0.0

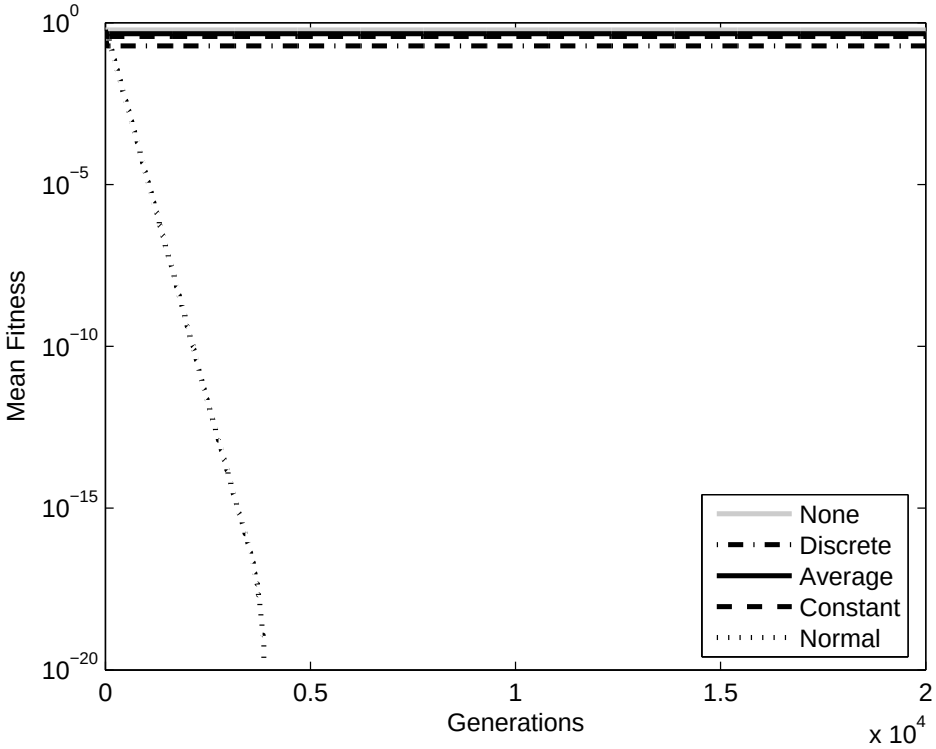

Figure 23. CGA results for SEL=None and MUT=None (24 samples each).

Table 10. Minimum fitness for SEL=None and Mut=Uniform (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	1.201e-3	1.760e-4
Discrete	1.254e-3	1.569e-4
Average	3.925e-4	7.280e-5
Constant	6.140e-4	5.169e-5
Normal	9.483e-3	1.186e-3

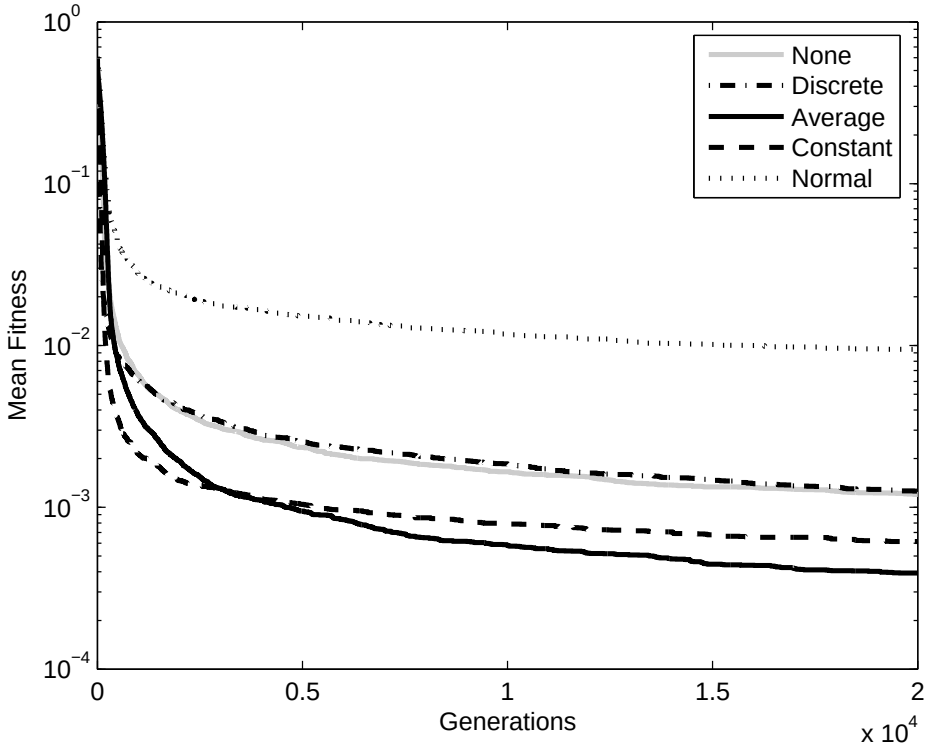

Figure 24. CGA results for SEL=None and MUT=Uniform (24 samples each).

Table 11. Minimum fitness for SEL=None and Mut=Normal (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	5.770e-4	1.080e-4
Discrete	6.368e-4	1.020e-4
Average	2.027e-4	5.023e-5
Constant	3.452e-4	4.811e-5
Normal	5.960e-3	6.111e-4

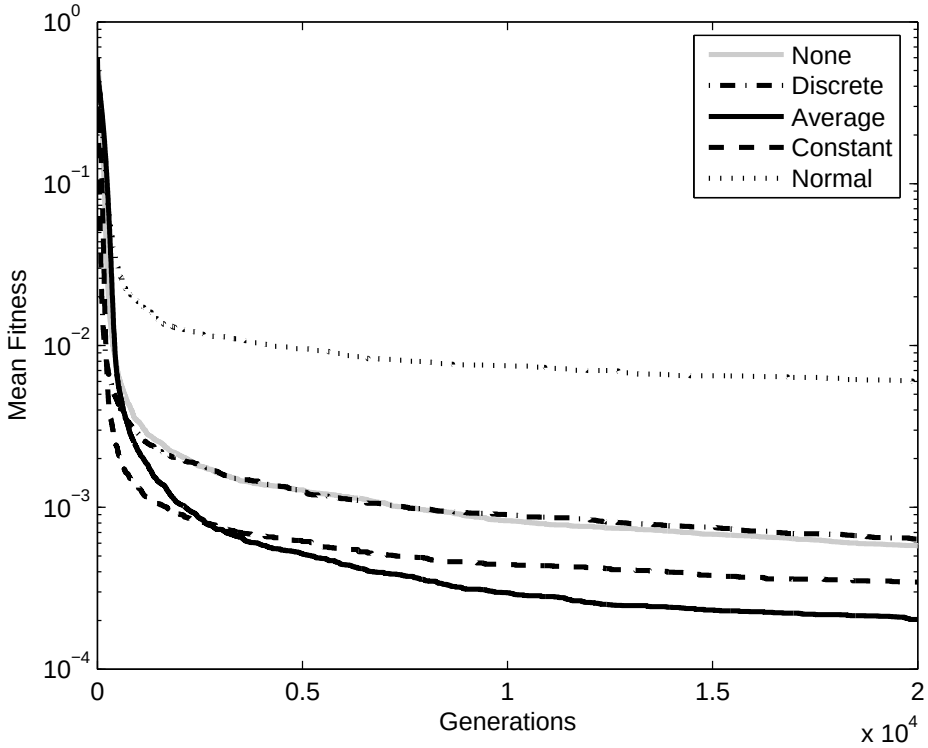

Figure 25. CGA results for SEL=None and MUT=Normal (24 samples each).

Table 12. Minimum fitness for SEL=None and Mut=BGA (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	1.191e-6	1.336e-7
Discrete	1.157e-6	1.437e-7
Average	8.708e-8	2.895e-8
Constant	2.976e-5	3.527e-6
Normal	1.001e-3	1.099e-4

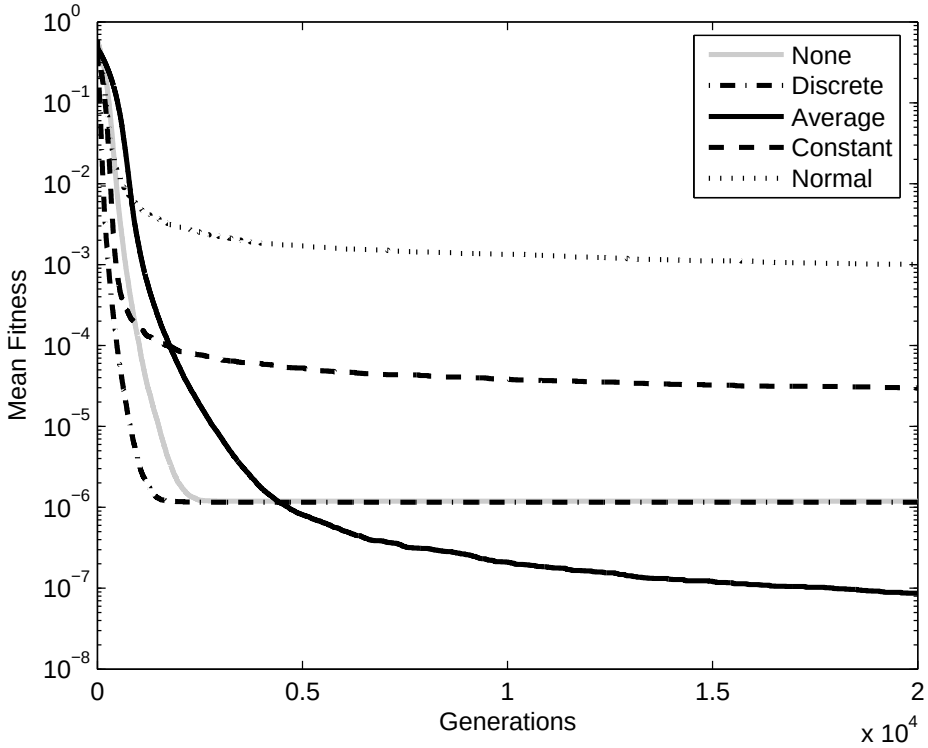

Figure 26. CGA results for SEL=None and MUT=BGA (24 samples each).

Table 13. Minimum fitness for SEL=Pro and Mut=None (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	5.797e-1	4.447e-2
Discrete	2.200e-1	3.261e-2
Average	4.410e-1	5.992e-2
Constant	3.462e-1	5.882e-2
Normal	2.776e-18	9.404e-18

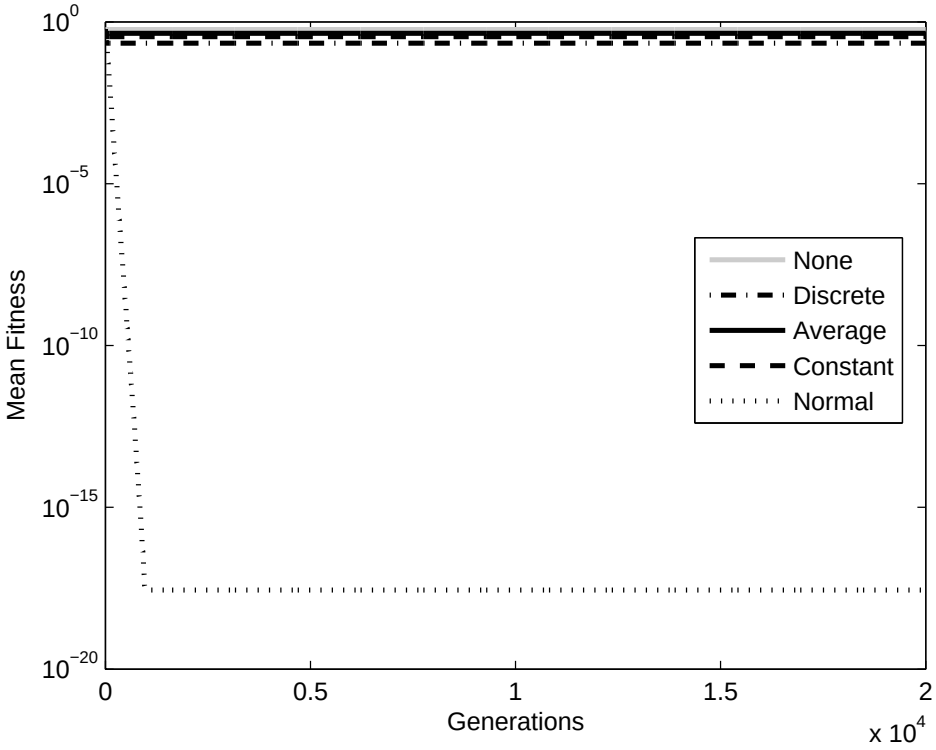

Figure 27. CGA results for SEL=Pro and MUT=None (24 samples each).

Table 14. Minimum fitness for SEL=Pro and Mut=Uniform (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	7.936e-4	1.218e-4
Discrete	7.915e-4	1.218e-4
Average	4.470e-4	9.508e-5
Constant	5.398e-5	9.376e-6
Normal	2.096e-17	2.257e-17

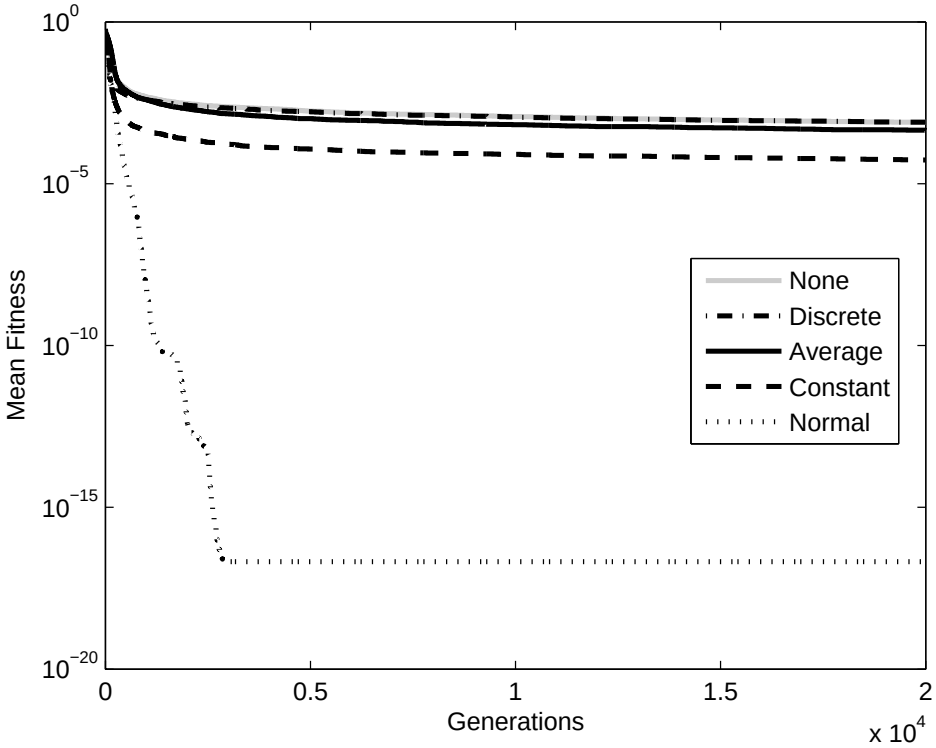

Figure 28. CGA results for SEL=Pro and MUT=Uniform (24 samples each).

Table 15. Minimum fitness for SEL=Pro and Mut=Normal (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	3.529e-4	5.501e-5
Discrete	3.932e-4	5.680e-5
Average	1.981e-4	5.124e-5
Constant	2.511e-5	7.001e-6
Normal	2.306e-17	3.218e-17

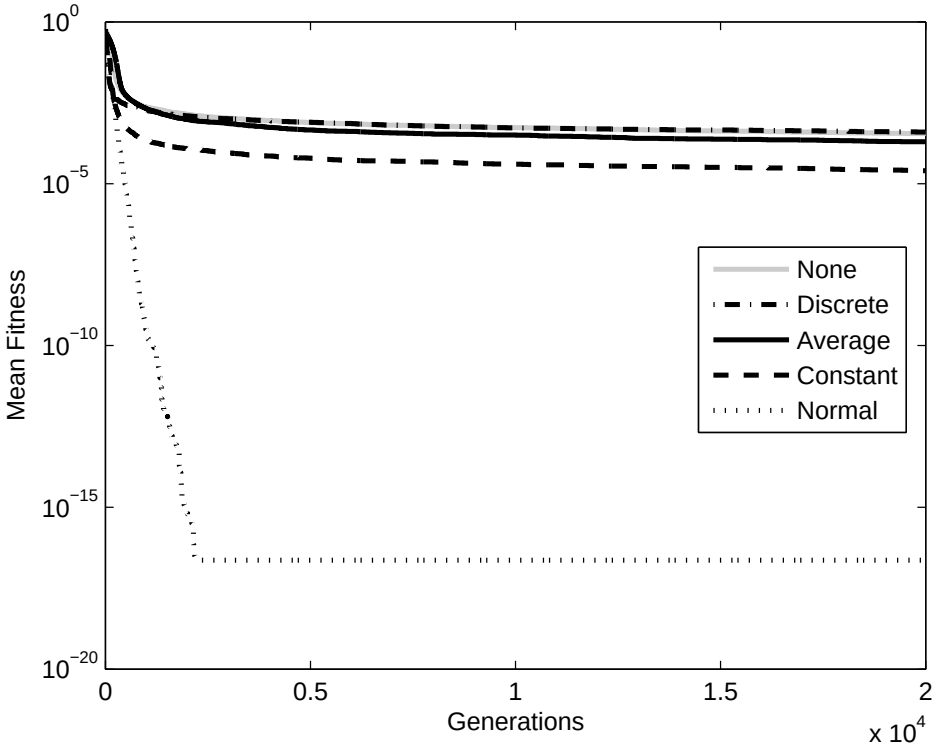

Figure 29. CGA results for SEL=Pro and MUT=Normal (24 samples each).

Table 16. Minimum fitness for SEL=Pro and Mut=BGA (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	1.169e-6	9.315e-8
Discrete	1.184e-6	1.719e-7
Average	1.063e-7	2.864e-8
Constant	1.043e-8	5.684e-9
Normal	1.712e-17	1.349e-17

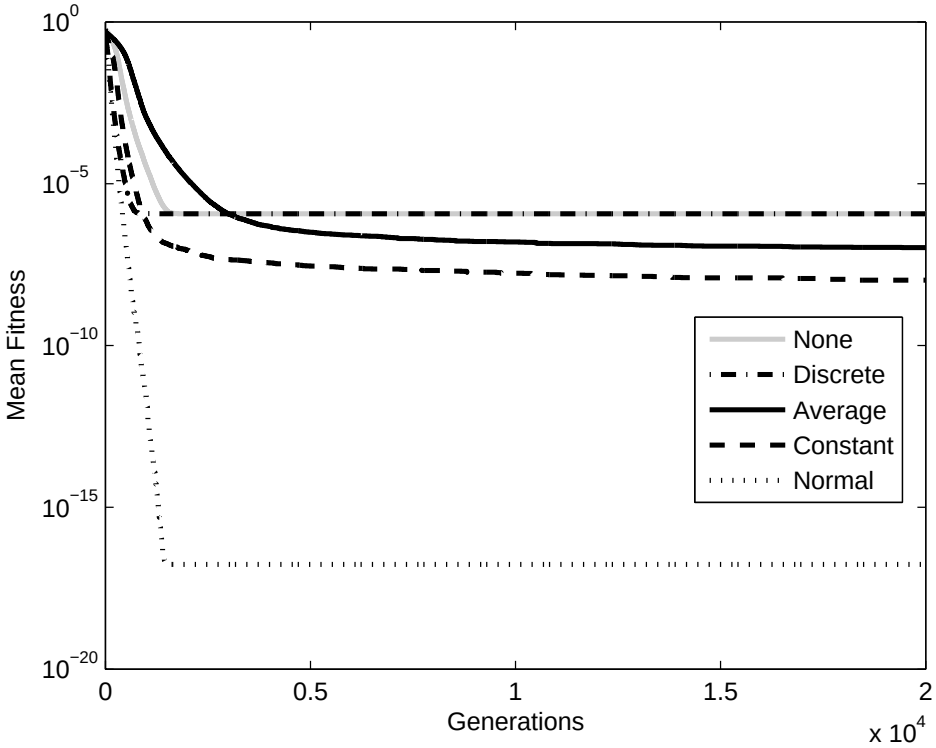

Figure 30. CGA results for SEL=Pro and MUT=BGA (24 samples each).

Table 17. Minimum fitness for SEL=Rank and Mut=None (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	5.897e-1	4.903e-2
Discrete	2.285e-1	4.435e-2
Average	4.279e-1	6.640e-2
Constant	2.660e-1	5.512e-2
Normal	6.014e-18	8.649e-18

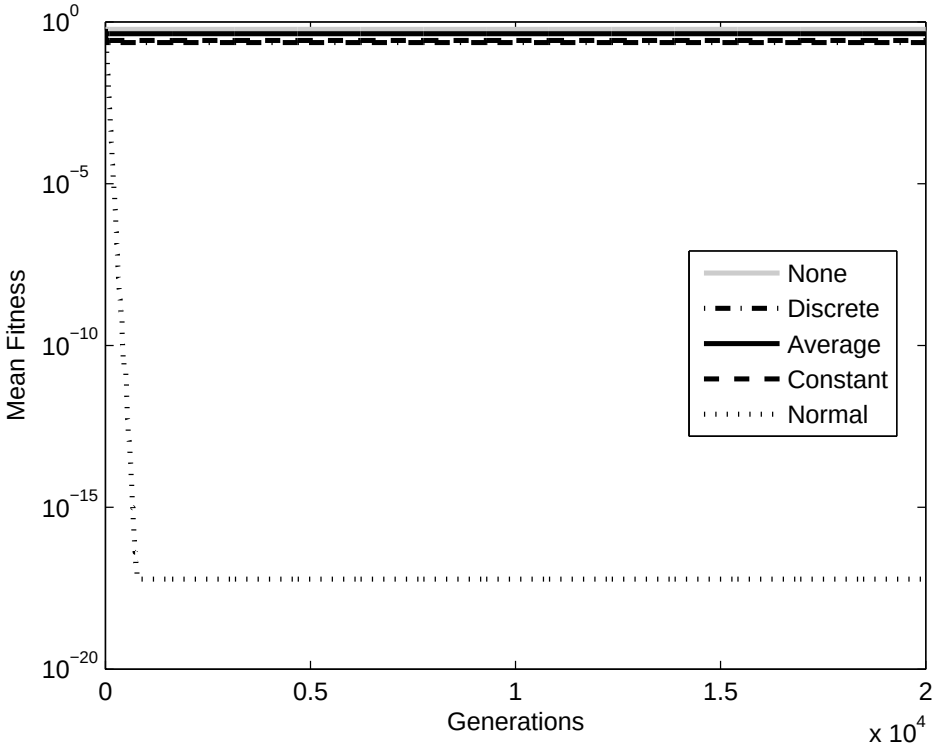

Figure 31. CGA results for SEL=Rank and MUT=None (24 samples each).

Table 18. Minimum fitness for SEL=Rank and Mut=Uniform (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	8.850e-4	1.697e-4
Discrete	8.334e-4	1.545e-4
Average	5.536e-4	1.333e-4
Constant	1.226e-6	3.793e-7
Normal	3.257e-7	3.372e-7

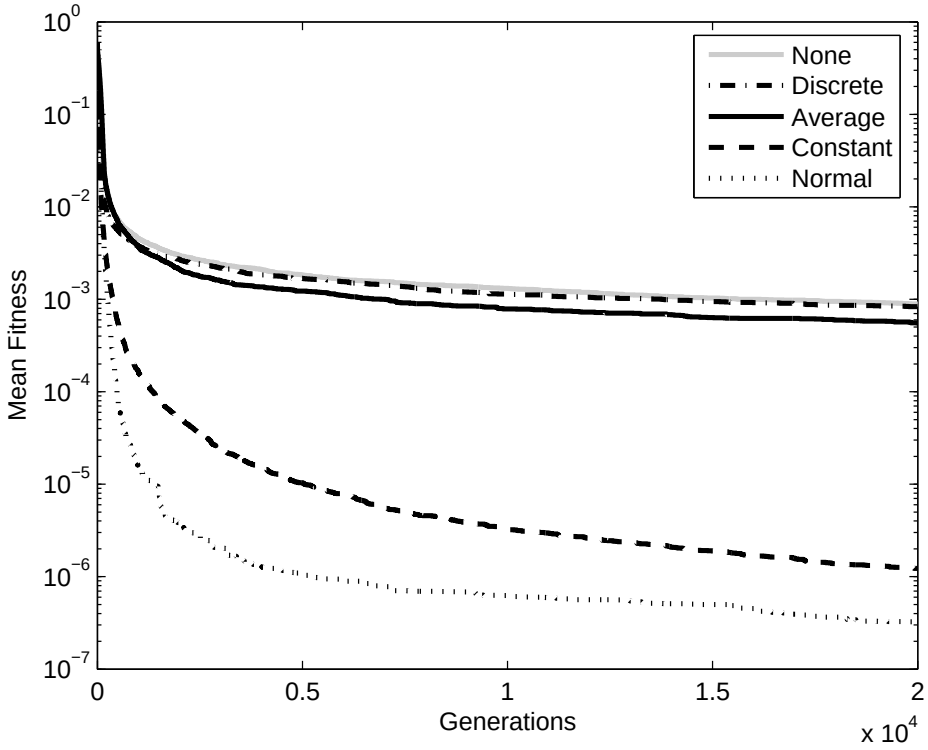

Figure 32. CGA results for SEL=Rank and MUT=Uniform (24 samples each).

Table 19. Minimum fitness for SEL=Rank and Mut=Normal (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	4.447e-4	8.244e-5
Discrete	3.814e-4	5.702e-5
Average	2.592e-3	5.805e-5
Constant	6.959e-7	3.643e-7
Normal	1.706e-7	9.988e-8

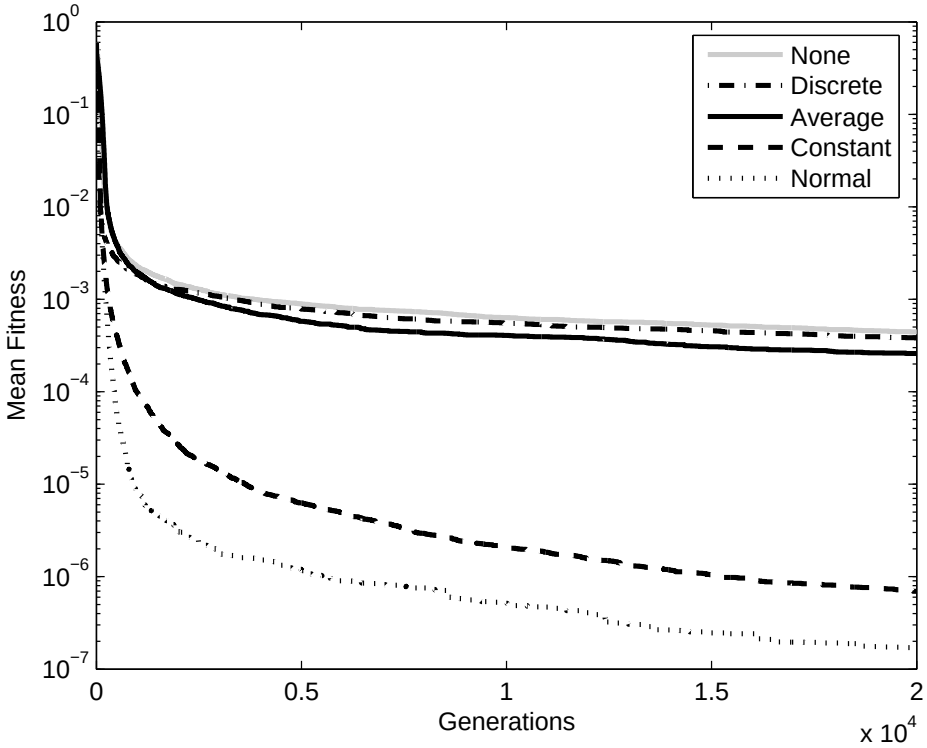

Figure 33. CGA results for SEL=Rank and MUT=Normal (24 samples each).

Table 20. Minimum fitness for SEL=Rank and Mut=BGA (24 samples each)

Crossover	**Mean**	**Std. Dev.**
None	1.165e-6	1.273e-7
Discrete	1.128e-6	1.130e-7
Average	1.623e-7	5.999e-8
Constant	2.910e-10	1.466e-10
Normal	3.460e-11	6.382e-11

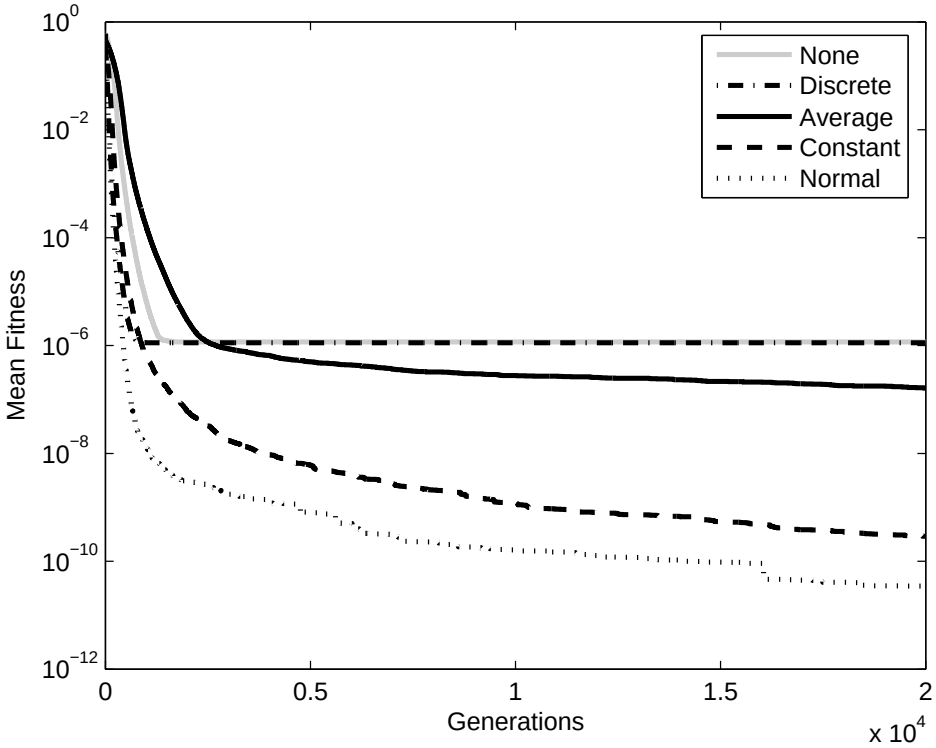

Figure 34. CGA results for SEL=Rank and MUT=BGA (24 samples each).

APPENDIX B. FDA MATLAB CODE

```
% FDA_RUN
%
% Filter Design Algorithm that uses a genetic algorithm   to design and
% optimizes an IIR transfer function that fits a desired magnitude
% response.   Adjustable inputs include the desired magnitude response
% (H_d), frequency specification points (W), frequnecy weighting values
% (Q), transfer fucntion order (alpha), population size (N), probability
% of crossover (p_c), maximimum number of generation executions
% (gen_max), minimum fitness stopping criteria (fit_min), and number of
% optimization attempts (attempt_max).
%
% SELECT_RANK, CROSS_NORMAL, and REPLACE_TRUNCATE functions are required by
% FDA_RUN.   Function FDA_ANALYZE is needed to analyze and plot the results.
%
% Author:  Mike Schmitz, 05-10-04
% North Dakota State University

% Housekeeping
clear
close
rand('state',sum(100*clock))       % create a random seed (VERY IMPORTANT!!!!)
tic
home

% directory path to store results in
save_path = '';         %'' equates to execution directory
% create a filename in which to store the FDA results
filename = 'data_raw_';

% define the desired magnitude response
H_d_description = '4th order Butterworth bandpass filter with w_l = pi/4, w_u = 3pi/4, and gain = 1';
% build vector W for which H_d is specified for
W = linspace(0,pi,10000);
% specify the desired magnitude response H_d
[z_d,p_d,K_d] = butter(2, [.25 .75]);
H_d = abs(K_d.*freqz(poly(z_d),poly(p_d),W));
% build vector Q for weighting the fitness equation
Q = ones(1,10000);
% specify the order of the desired transfer function
alpha = 4;

% number of elements in population
N = 200;
% number of vectors per element
M = alpha;
% probability of crossover
p_c = .7;
% max number of generations per problem
gen_max = 2000;
% min fitness level
fit_min = 0;
% number of design attempts
attempt_max = 5;
```

```
% create a txt file for storing a description of the desired filter
fid = fopen([save_path,'README_.txt'],'w');
fprintf(fid, 'Desired Filter Specs...\n');
fprintf(fid,['Description:  ',H_d_description,'\n']);
fprintf(fid,['alpha:  ',num2str(alpha),'\n']);
fprintf(fid,['N: ',num2str(N),'\n']);
fprintf(fid,['M: ',num2str(M),'\n']);
fprintf(fid,['p_c:  ',num2str(p_c),'\n']);
fprintf(fid,['gen_max:  ',num2str(gen_max),'\n']);
fprintf(fid,['fit_min:  ',num2str(fit_min),'\n']);
fprintf(fid,['attempt_max:  ',num2str(attempt_max),'\n']);
fclose(fid);

% create a sturcture for data storage
archive=struct('elements',[],'fitness',[]);

for a=1:attempt_max
       % Display the attempt in progress
       disp(['Attempt = ',num2str(a)]);

       % create the initial population of filters by locating poles and zeros
       % with a uniform PDF in the unit circle
       mag = sqrt(rand(N,alpha));
       ang = pi*rand(N,alpha);

       % build a structure for handling the population of elements,
       % corresponding fitness, the selection subset of elements, and
       % generated offspring elements
       gen.pop.elements = mag.*exp(j.*ang);
       gen.pop.fitness = [];
       gen.selection = [];
       gen.offspring = [];

       % Start the FDA
       for g = 1:gen_max
              % Evaluate fitness of current generation
              for n=1:N
                     % get the zeros and poles of x_n
                     z_n = cat(2,gen.pop.elements(n,1:alpha/2),conj(gen.pop.elements(n,1:alpha/2)));
                     p_n = cat(2,gen.pop.elements(n,(alpha/2)+1:end),conj(gen.pop.elements(n,(alpha/2)+1:end)));
                     %calculate the magnitude response of H_n
                     H_n = abs(freqz(poly(z_n),poly(p_n),W));
                     % scale H_n by K_n
                     K_n = sum(H_d,2)./sum(H_n,2);
                     H_n = K_n.*H_n;
                     % calculate magnitude squared error
                     e_2 = abs(H_n - H_d).^2;
                     % calculate fitness
                     gen.pop.fitness(n,1) = (1./length(W)).*(Q*e_2');
              end

              % check for fit_min stopping criteria
              best = min(gen.pop.fitness,[],1);
              if(best <= fit_min)
```

```
                disp('Minimum fitness achieved.');
                break;
            end

            % store element and fitness results in archive structure
            archive(a,g).elements=gen.pop.elements;
            archive(a,g).fitness=gen.pop.fitness;

            % perform selection
            gen.selection = select_rank(gen);
            % perform hybrid crossover
            gen.offspring = cross_normal(gen,p_c);

            % find poles and zeros outside the unit circle
            out = abs(gen.offspring) > 1;
            % reciprocate necessary poles and zeros
            gen.offspring = (1./gen.offspring).*out + gen.offspring.*abs(1-out);

            % perform replacement
            gen.pop.elements = replace_truncate(gen);
        end %g

        % Display time to complete last attempt
        disp(['Time = ',num2str(toc),' seconds']);
end %attempt

% calculate simulation time for experiment
time = toc;

% Save all necessary data and results to disk for later analysis
save([save_path,filename,'.mat'],'H_d','K_d','z_d','p_d','W','Q','archive','time');

disp('Design Complete!');
disp(['Time = ',num2str(time),' seconds']);
```

```
 function selection = select_rank(gen)
% SELECT_RANK performs rank-based selection.
% SELECTION = SELECT_RANK(GEN) selects elements from GEN
% for recombination using linear scaled rank-based selection.
% The selected elements are returned in SELECTION.
%
% Author:  Mike Schmitz, 04-19-04
% North Dakota State University

% Concatonate gen.pop.elements and gen.pop.fitness
[er,ec] = size(gen.pop.elements);
[fr,fc] = size(gen.pop.fitness);
a = cat(2,gen.pop.elements,gen.pop.fitness);

% Sort rows of 'a' in ascending order according to fitness
a = sortrows(a,(ec + fc));

% Reverse the order so it is now decending
a = flipud(a);

% Calculate the probability of selecting an individual
ps = [1:fr]'./sum([1:fr]);

% Create a running sum probability vector
ps = cumsum(ps);

% Expand into an array with the number of columns equal to the
% number of elements in the population.   All columns are identical.
ps = ps*ones(1,er);

% Create a square array of random numbers with all rows identical.
r = ones(fr,1)*rand(1,er);

% Use a conditional to select individuals based on ps and r
selection = (cumsum(ps > r) == ones(size(ps)))'*a(:,1:ec);
```

```
 function offspring = cross_normal(gen,p_c)
% CROSS_NORMAL performs normal crossover.
% OFFSPRING = CROSS_NORMAL(GEN,P_C) performs normal crossover with
% probability P_C to the elements in GEN.SELECTION.   The resulting
% offspring is returned in OFFSPRING.
%
% Author:  Mike Schmitz, 04-19-04
% North Dakota State University

[sr,sc] = size(gen.selection);

% Choose elements for crossover
parents = [];
for i=1:sr
       if (rand(1) < p_c)
               parents(end+1,:)  = gen.selection(i,:);
       end
end

[pr,pc] = size(parents);

% Split parents into two groups
len = floor(pr/2);
parent_1 = parents(1:len,:);
parent_2 = parents(len+1:2*len,:);

% mid = (x_1 + x_2)/2
mid = (parent_1 + parent_2)./2;

% f(x) = N[0,.741]
X = (.741)*randn(size(parent_1));

% Re(x_3) = Re(mid) + X*Re(x_2-x_1)
% Im(x_3) = Im(mid) + X*Im(x_2-x_1)
Re = real(mid) + X.*real(parent_2 - parent_1);
Im = imag(mid) + X.*imag(parent_2 - parent_1);

% x_3 = Re(x_3) + j*Im(x_3)
offspring = Re + j*Im;

% f(x) = N[0,.741]
X = (.741)*randn(size(parent_1));

% Re(x_4) = Re(mid) + X*Re(x_2-x_1)
% Im(x_4) = Im(mid) + X*Im(x_2-x_1)
Re = real(mid) + X.*real(parent_2 - parent_1);
Im = imag(mid) + X.*imag(parent_2 - parent_1);

% x_4 = Re(x_4) + j*Im(x_4)
offspring(len+1:2*len,:)  = Re + j*Im;
```

```
 function gen_next = replace_truncate(gen)
% REPLACE_TRUNCATE performs truncation replacement.
% GEN_NEXT = REPLACE_TRUNCATE(GEN) forms the next generation
% population of elements GEN_NEXT by removing the least fit
% elements from population GEN and inserting the offspring
% elements.
%
% Author:  Mike Schmitz, 04-19-04
% North Dakota State University

[or,oc] = size(gen.offspring);
[pr,pc] = size(gen.pop.elements);
[fr,fc] = size(gen.pop.fitness);

% Concatonate the gain, zeros, poles, and fitness array
a = cat(2,gen.pop.elements,gen.pop.fitness);
% Sort rows of 'a' in ascending order according to fitness
a = sortrows(a,(pc + fc));
% replace least fit with the offspring
gen_next = cat(1,a(1:pr-or,1:pc),gen.offspring);
```

```
% FDA_ANALYZE
%
% Takes the data results from FDA_RUN and parses out the best filter design
% based on fitness.   The transfer function (K_n, z_n, p_n) and magnitude
% response (H_n) of the best element (x_n) are calculated and saved to disk
% along with the minimum fitness value (fit_min) for each generation.
% Plots are generated and saved to disk for the pole-zero plot of H_n,
% magniude response comparison betwenn H_n and H_d in both amplitude and
% dB, and the minimum fitness vs generation curve for analyzing convergence
% rate.
%
% Author:  Mike Schmitz, 05-10-04
% North Dakota State University

% housekeeping
clear
close
tic
home

% directory from which to load data
load_path = '';
% directlry for storing results of analysis
save_path = '';
% file where the raw results from FDA_RUN are stored
filename = 'data_raw_';

% load the data generated by FDA_RUN
load ([load_path,filename,'.mat']);

[attempt_max,gen_max]=size(archive);

% find the minimum fitness for every generation of every attempt
for a=1:attempt_max
       for g=1:gen_max
              archive_fit_min(a,g)=min(archive(a,g).fitness,[],1);
       end
end

% find the minimum overall fitness
temp = min(archive_fit_min,[],1);
overall_fit_min = min(temp,[],2);

% find the attempt and gen for overall minimum fitness value
[attempt_min,gen_min] = find(archive_fit_min==overall_fit_min);

% in case of multiple matches, use the first attempt and last gen
attempt_min = attempt_min(1);
gen_min = gen_min(end);

% find x_n for overall_fit_min
[N,M] = size(archive(attempt_min).elements);
ef = cat(2,archive(attempt_min).elements,archive(attempt_min,gen_min).fitness);
ef = sortrows(ef,M+1);
```

```
x_n = ef(1,1:M);

% calculate z_n, p_n, K_n, and H_n form x_n
alpha = length(x_n);
z_n=cat(2,x_n(1:alpha/2),conj(x_n(1:alpha/2)));
p_n=cat(2,x_n((alpha/2)+1:alpha),conj(x_n((alpha/2)+1:alpha)));
%calculate the magnitude response of H_n
H_n = abs(freqz(poly(z_n),poly(p_n),W));
% scale H_n by K_n
K_n = sum(H_d,2)./sum(H_n,2);
H_n = K_n.*H_n;

% get the minimum fitness vs gen data for the attempt_min
fit_min = archive_fit_min(attempt_min,:);

% save the data for the best (minimum) attempt
save([save_path,'data_min_','.mat'],'x_n','H_n','K_n','z_n','p_n','fit_min','H_d','W','Q');

% plot the poles and zeros of H_n (designed filter)
zplane(z_n',p_n');
grid on;
% save the current plot as a .fig and .eps
saveas(gca,[save_path,'polezero_H_n_','.fig']);
print('-depsc',[save_path,'polezero_H_n_'])


% plot the magnitude response of H_n and H_d
semilogx(W,H_d,'b',W,H_n,'r');
grid on;
xlabel('\Omega');
ylabel('Magnitude');
legend('Desired','Designed',3);
% save the current plot as a .fig and .eps
saveas(gca,[save_path,'magnitude_','.fig']);
print('-depsc',[save_path,'magnitude_'])

% plot the magnitude (dB) response of H_n and H_d
semilogx(W,20.*log10(H_d),'b',W,20.*log10(H_n),'r');
grid on;
xlabel('\Omega');
ylabel('Magnitude (dB)');
legend('Desired','Designed',3);
% save the current plot as a .fig and .eps
saveas(gca,[save_path,'decibel_','.fig']);
print('-depsc',[save_path,'decibel_'])

% plot fit_min vs generation
semilogy(fit_min);
grid on;
xlabel('Generations');
ylabel('Fitness');
% save the current plot as a .fig and .eps
saveas(gca,[save_path,'fitness_','.fig']);
print('-depsc',[save_path,'fitness_'])
```

```
disp('Analysis Complete!');
disp(['Time = ',num2str(toc),' seconds']);
```